UCLouvain

epl

École polytechnique de Louvain

OpenMRS as a backend for multi-centric clinical research studies

Author: **Thierry Guiot**
Supervisor: **Sébastien Jodogne**
Readers: **Charles Pecheur, Donatien Schmitz**
Academic year 2025–2026
Master [120] in Computer Science

Abstract

Multi-centric clinical research is essential for generating robust medical evidence. But it faces obstacles in certain resource-constrained environments. While Electronic Data Capture (EDC) systems like REDCap are standard in well-resourced settings, their deployment alongside existing Electronic Health Record (EHR) systems implies infrastructure costs, maintenance burdens, and the complexity of managing parallel systems. OpenMRS, a widely adopted open-source EHR platform in sub-Saharan Africa and Asia, currently lacks native support for standardised research data collection.

This master's thesis presents the design and implementation of "Fhirquestionnaires," an OpenMRS module that integrates standardised, FHIR-compliant research data collection natively within the OpenMRS ecosystem. By leveraging the "Questionnaire" and "QuestionnaireResponse" HL7 FHIR resources, the module enables research coordinators to create, import, and distribute structured research questionnaires without requiring external infrastructure.

Furthermore, it exposes a RESTful interoperability API based on HAPI FHIR, allowing seamless data exchange with third-party systems.

An end-to-end proof of concept demonstrates the module's capability to support the full lifecycle of a multi-centric study, from questionnaire design at a coordinating centre to distributed data collection and aggregated export. By eliminating the need for separate EDC systems, this work offers an open-source solution that enhances data quality and interoperability for clinical research in resource-limited settings, contributing to a more equitable global research infrastructure.

Acknowledgements

First of all, I would like to express my warmest thanks and gratefulness to my supervisor, Professor Sébastien Jodogne, for his support and assistance throughout the course of this thesis. His kindness, his unfailing availability, his attentive listening and his valuable advice have been of great help to me.

I would also like to thank Professor Jodogne's entire team (including PhD students but also other students working on their master's thesis) for our exchanges, their advice and the fruitful discussions we had both during and outside our weekly meetings.

Finally, I would like to express my deepest gratitude to my parents, family, friends and "cokoteurs" for their unfailing support, not only during this project but also throughout my studies and my life in general.

Contents

List of abbreviations

AI Artificial Intelligence. 30

API Application Programming Interface. 9, 14, 16–18, 20–23, 26, 28, 29, 31, 33, 34, 36–38, 40, 42, 44, 46, 48, 49, 51

CRUD Create, Read, Update, Delete. 20, 34

CSS Cascading Style Sheets. 30, 34

CSV Comma-Separated Values. 12, 19

DAO Data Access Object. 23

EDC Electronic Data Capture. 7–9, 12, 13, 18, 51, 54

EHR Electronic Health Record. 8, 12, 13

EMR Electronic Medical Record. 7, 13

FHIR Fast Healthcare Interoperability Resources. 8, 9, 11, 15–28, 30–32, 34–40, 42–46, 48, 49, 51–54

GSP Groovy Server Pages. 31, 33, 34

HIV Human Immunodeficiency Viruses. 7, 13

HL7 Health Level Seven. 8, 11, 15

HRQOL Health-Related Quality Of Life. 42

HTML Hypertext Markup Language. 22, 28, 30, 32, 34, 39, 40, 45

HTTP Hypertext Transfer Protocol. 16, 32

IT Information Technology. 7, 12, 13

JSON JavaScript Object Notation. 16, 19, 21, 24–28, 30–32, 35–40, 42–44, 46, 49, 51, 52

JSP JavaServer Pages. 14

MVC Model-View-Controller. 14, 22

REST Representational State Transfer. 16, 17, 20, 22, 28, 31, 33, 34, 37, 39, 40, 49, 51

SDK Software Development Kit. 16, 33

SQL Structured Query Language. 24–26

UI User Interface. 21, 23, 26, 27, 30, 34, 36–38, 45

URL Uniform Resource Locator. 46

UUID Universally Unique Identifier. 26, 29, 32, 36–38, 43, 44, 48, 49

XML Extensible Markup Language. 16, 31, 32

Introduction

Clinical research drives medicine forward by providing evidence. However, the reliability of this evidence depends on the reliability of the data on which it is based. In multi-centric clinical trials, where a single protocol is implemented across several hospitals or clinics, the challenge of generating clean, consistent and comparable data is real. Each participating site brings its own patient population, clinical practices and information systems. Harmonising data across this diversity is not a secondary concern: it is one of the key factors determining whether a multi-centric study produces generalisable results or a set of incomparable local measurements.

The tools used to collect research data are therefore critical infrastructure. Electronic Data Capture (EDC) systems (specialised software programs for designing data collection instruments, recording patient responses, and exporting structured datasets) have become the standard approach in well-resourced research environments. Tools like REDCap, OpenClinica, and Castor EDC have been widely adopted by academic medical centres.

Yet a significant volume of multi-centric studies are conducted in hospitals and clinics across sub-Saharan Africa and Southeast Asia. These institutions sometimes face a specific set of constraints: intermittent power and connectivity, limited local technical capacity, and tightly constrained budgets. In these contexts, the infrastructure assumptions embedded in most EDC tools (like a permanent internet connection, a dedicated IT team, a licensing fee) may undermine the organisation of such studies.

The OpenMRS Ecosystem and Its Gap

OpenMRS is an open-source Electronic Medical Record (EMR) system developed specifically to address the needs of healthcare in resource-constrained environments. Launched in response to the HIV crisis in sub-Saharan Africa, it has since expanded to support the care of millions of patients across over forty countries.

Despite its widespread deployment in precisely the settings where clinical research is needed and most difficult to conduct, OpenMRS lacks native support for standardised research data collection. Clinicians and researchers using OpenMRS who wish to run a multi-centric study currently face an uncomfortable choice: either add a separate EDC system alongside OpenMRS (doubling infrastructure, maintenance, and training burdens) or work around the absence of dedicated tooling with improvised solutions. Neither option is satisfactory. The first is unsustainable for resource-constrained sites and the second undermines the data quality that makes multi-centric research valuable.

There is, however, a potential path through this gap. The Health Level Seven (HL7) Fast Healthcare Interoperability Resources (FHIR) standard provides a rich, well-specified data model for representing clinical questionnaires and their responses. Its "Questionnaire" and "QuestionnaireResponse" resources define exactly the kind of structured, portable, and interoperable data collection instruments that multi-centric research requires. FHIR has been widely adopted in national digital health programmes and commercial Electronic Health Record (EHR) systems, and an active ecosystem of open-source libraries and tools has grown around it. The question this master's thesis addresses is whether FHIR-based research data collection can be built natively into OpenMRS, as a module that requires no additional infrastructure, and exposes a standard interface that allows questionnaires and responses to be exchanged between sites and systems.

Research Objective and Contribution

This master's thesis presents the design and implementation of **Fhirquestionnaires**: an OpenMRS module that enables FHIR-compliant research data collection natively within the OpenMRS platform. The module allows a research coordinator to create or import a structured questionnaire using the FHIR "Questionnaire" resource, distribute it to participating sites, collect responses within the patient record, and export the resulting dataset for analysis. Without requiring any external EDC system or modifications to the OpenMRS core.

This work makes a triple contribution. First, it addresses a concrete and significant gap in the OpenMRS ecosystem: the absence of a standardised, interoperable research data collection capability. Second, it demonstrates that FHIR resources can be stored and served efficiently within OpenMRS's existing architecture through a wise design strategy. Third, it produces an open-source, deployable module that can be used, extended, and maintained by the OpenMRS community.

Scope and Limitations

Before diving into the details, it is important to be clear about what this work does and does not attempt. The Fhirquestionnaires module is designed for the OpenMRS O2 platform, which remains the most widely deployed. It is not designed for the newer O3 platform. Migration to O3 is a natural direction for future work, but falls outside the scope of this project.

The module supports the creation, distribution, collection, and export of FHIR R4 "Questionnaire" and "QuestionnaireResponse" resources. However, it does not implement the full FHIR R4 specification: several optional features of the "Questionnaire" resource (such as version management and copyright metadata) are not currently supported via the provided graphical interface. Indeed, the design prioritises ease of use and deployability over feature completeness.

The proof-of-concept provided in chapter 4 demonstrates the module's functionalities through an end-to-end scenario. It does not include a formal user study, a performance evaluation, or a deployment in a real clinical environment.

These are recognised limitations of the current work, discussed further in the conclusion.

Structure

This master's thesis is organised into four main chapters, preceded by this introduction and followed by a conclusion.

Chapter 1 reviews the challenges of multi-centric clinical research, the landscape of existing EDC solutions and their limitations in OpenMRS contexts, the architecture of OpenMRS itself, and the FHIR standard and its relevance to form-based research data collection.

Chapter 2 presents the design and architecture of the Fhirquestionnaires module. It begins with a requirements analysis, then develops the overall system architecture, the database design, the user interface layout, and the interoperability Application Programming Interface (API) specification. Each design decision is motivated by the constraints and requirements established in chapter 1.

Chapter 3 describes the implementation of the module in detail, including its technology stack, the project structure, and the implementation of a few key features in both the backend and the frontend.

Chapter 4 demonstrates the system through an end-to-end scenario that follows the full lifecycle of a multi-centric research questionnaire, from creation and distribution to data collection, export and analysis.

The conclusion synthesises the findings, reflects on the limitations of the current work, and proposes directions for future development.

Technical deliverable

The source code of the module is licensed under the *Mozilla Public License*, version 2.0[1] and is available at `https://github.com/guiott67/Fhirquestionnaires`.

A demo video can also be found at `https://www.youtube.com/watch?v=vOBslBC2Ws0`.

[1] `https://www.mozilla.org/en-US/MPL/2.0/`

Chapter 1

State of the Art

Multi-centric clinical research depends on the systems that collect, harmonise, and exchange data across sites. Before designing any new tool, it is essential to understand the landscape (what already exists and what is still lacking today). This chapter addresses that matter. It begins by discussing the specific challenges of multi-centric data collection, then reviews the most commonly used electronic data capture solutions. It then turns to OpenMRS and concludes with an overview of the HL7 FHIR standard, whose data model and interoperability principles will form the basis for the module developed in the next chapters.

Together, these four apsects define the problem space that the rest of this master's thesis addresses.

1.1 Multi-centric Clinical Research

As a cornerstone of clinical and public health research (particularly for Phase II-III randomised controlled trials assessing efficacy and safety of new treatments [1], [2]), multi-centric clinical studies implement a single, standardised protocol at multiple sites. They rely on a coordinating center responsible for uniform implementation and harmonised outcome assessment [3], [4].

In contrast to single-centre studies, this model enables larger cohorts to be recruited and enhances the statistical power of the results by spanning multiple settings and populations. This can improve generalisability to heterogeneous populations [5].

This heterogeneity drives scientific value but also introduces considerable complexity. Differences in patient profiles, local practices, data systems and regulatory environments result in heterogeneous and fragmented data [3].

Therefore, successful and relevant multi-centre studies strongly rely on good coordination and high-quality data harmonisation [6].

1.2 Existing electronic data capture solutions

In response to the challenges of standardised data collection in clinical research, several categories of tools have emerged: research-specific EDC and care-oriented EHR systems. A comparison of these reveals that they complement one another, but also highlights significant limitations in certain contexts.

1.2.1 REDCap: the leading academic EDC software

REDCap, which stands for Research Electronic Data Capture, is a web application developed and maintained by Vanderbilt University, designed for the creation and management of research databases and surveys [7]. Indeed, it allows researchers to collect, store, and analyse data securely through a survey based approach. Researchers define their data collection instruments (forms) by specifying attributes like field labels, field types (e.g., text, date, dropdown), and validation rules in a user-friendly interface or a simple data dictionary (often a CSV table) [8]. Its business model is based on a consortium of partner institutions: the tool is available for free to academic and non-profit organisations that are members of the consortium, but its commercial use is subject to contractual terms.

REDCap also stands out for its strong training ecosystem (e.g. workshops, consortium community) and variety of integrated educational resources (like video tutorials and context-sensitive documentation), which gives it a decisive advantage in terms of adoption by non-IT researchers [9].

1.2.2 Other EDC tools

OpenClinica is web-based, open-source EDC software. Designed for large-scale clinical trials, it offers modules for study setup, data submission, monitoring, and extraction. And also supports multi-centric trials with advanced features for site management [10]. Its flexibility makes it a powerful tool that has been successfully used for large multi-centric trials [11]. However, it still requires substantial setup and validation, which can be challenging for many low-resource settings [12].

Castor EDC offers a balance between easy-to-use native mobile apps and structured clinical trial management. Although its official pricing is not publicly available, costs are likely to depend on the scope of the study and could to present a major obstacle for smaller organisations with limited resources [13].

Recent work by Jakob Vielhauer et al. [14] proposes a very lightweight, open-source EDC written in R, explicitly contrasting its minimal deployment needs with the more complex installation and customisation requirements of OpenClinica and proprietary systems such as REDCap for institutions lacking advanced IT skills or consortium access.

1.2.3 Constraints in OpenMRS contexts

The solutions described above share an implicit assumption: they operate in parallel with the EHR system, requiring double data entry or specific technical integration. In well-resourced environments, this integration is easily feasible.

However, in the context of OpenMRS, which is deployed primarily in resource-constrained countries in sub-Saharan Africa and Asia, the situation is different. Barriers include high costs of procurement and maintenance, sometimes unreliable power, and limited local technical capacity [15]–[17].

Duplicating systems by using separate EDC programs such as REDCap or OpenClinica adds infrastructure, maintenance and training burdens that local teams might be unable to sustain. The central question is therefore whether it is possible to natively integrate standardised clinical research functionality into systems like OpenMRS, without the need of an external EDC system.

1.3 OpenMRS: an open-source EHR

1.3.1 Origins and global roll-out

OpenMRS is an open-source, electronic medical record platform launched in 2004 in response to the health crisis surrounding HIV care in sub-Saharan Africa [18]. Its objective is to provide a general-purpose, robust, adaptable and accessible EHR in resource-constrained environments. Early deployments in Kenya, Rwanda and South Africa in 2006 demonstrated that a general-purpose, configurable EMR could be adapted to new sites, languages and diseases in low-resource settings. Since then, the system has expanded to over 40 countries, mainly in Africa and Asia. Nowadays, OpenMRS supports healthcare for more than 12 million people,

and this figure continues to grow [19]. One of its key advantages is its flexibility of deployment: OpenMRS can be hosted on anything from a simple laptop to an enterprise server, on Linux, Windows or MacOS. Making it accessible even in environments with limited technical infrastructure.

1.3.2 Modular architecture: core and extensions

The architecture of OpenMRS is based on a fundamental principle: the separation between a stable core and optional modules that extend its functionality without altering the core code. This approach ensures the stability of the system and allows for a high degree of adaptability to local needs [18].

The backbone of OpenMRS is its central API. This API provides methods for all core functions: adding or updating a patient, an encounter, an observation... These methods are provided within service classes: PatientService, ConceptService, EncounterService, ObsService...

The architecture is divided into three distinct layers:

- User interface layer: Built using Spring Model-View-Controller (MVC), JavaServer Pages (JSP) and JavaScript. Note that it has recently evolved to version O3, which uses React instead.
- Service Layer: Orchestrated by the Spring framework with dependency injection, it contains the business logic. The *Context* is a static class that maintains a single instance of each service (PatientService, EncounterService, etc.).
- Data Access Layer: Built on Hibernate as an Object-Relational Mapping tool, and Liquibase for managing schema migrations independently of the database. This layer abstracts the actual data model.

OpenMRS features a custom module framework that allows the core functionality to be extended and modified. Modules are standalone components that can be enabled or disabled dynamically, without requiring a full system restart.

1.3.3 Data model: key concepts

An understanding of the OpenMRS data model is essential for designing new modules. Here is an overview of a few key entities of this model:

Concept: OpenMRS is based on a "concept dictionary" that describes all the data elements that can be stored in the system: clinical findings, laboratory test results, etc. This approach avoids the need to modify the database structure to add new conditions and facilitates the sharing of data dictionaries between projects and sites. A concept therefore represents the semantic definition of what is observed, regardless of its value.

Patient: It is the central component of the system. It contains demographic data and identifiers. It serves as the anchor point for the entire medical record.

Visit: Represents a period of interaction between the patient and the healthcare facility (e.g. a hospital stay, a series of related consultations). A visit can contain several encounters.

Encounter: Represents a specific clinical interaction (consultation, examination, dispensing of medication).

Observation (Obs): This is the system's most granular data unit. Each Obs links a patient to a concept (via its value), within the context of an encounter. For example: 'Systolic blood pressure = 104.8 mmHg'.

1.4 FHIR: A standard for healthcare interoperability

1.4.1 What is FHIR?

FHIR, which stands for *Fast Healthcare Interoperability Resources*, and is developed by HL7 International [1], is a standard for the exchange of health information. Its aim is to provide most of the core functionality that is needed to represent data in healthcare contexts and exchange it between different systems. FHIR provides a logical data model for representing the entities involved in various types of healthcare transactions [20]. It builds on lessons from earlier standards like HL7 v2 and v3, while leveraging modern web technologies to simplify implementation and promote interoperability [21], [22]

The FHIR approach is based on the concept of "resources". They are modular building blocks that represent units of clinical or administrative information (Patient, Observation, Medication, Encounter, Questionnaire, etc.). These resources

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

can be combined or linked via references to address specific clinical and administrative workflows.

From a technical perspective, FHIR uses modern web technologies: RESTful [2] APIs over HTTP with JavaScript Object Notation (JSON) and Extensible Markup Language (XML) serialisation.

1.4.2 Why is FHIR gaining popularity over other standards?

Compared to older standards like HL7 v2/v3, FHIR enables a less complex implementation thanks to REST, JSON and XML, as well as widely available tools and SDKs.

As explained by Vorisek, Lehne, Klopfenstein, *et al.* [21], another of the biggest advantages offered by FHIR is its extensibility and profiling mechanisms that can be used to adapt base resources to local or domain-specific needs while preserving interoperability.

These two key advantages largely explain its worldwide adoption within national programmes and digital health ecosystems.

1.4.3 FHIR resources relevant to form-based research

For use cases involving the collection of research data using forms and surveys, several FHIR resources are particularly relevant:

Patient: a core resource containing demographic data (identity, date of birth, gender, contact details).

Questionnaire: a definitional resource representing a structured set of questions intended to standardise the data collection process (forms, surveys, assessments). It defines the content, order, and constraints of questions but is subject-independent and does not contain actual answers.

QuestionnaireResponse: a record resource capturing the specific answers provided to a Questionnaire by a subject (patient, organisation, practitioner, device...). It mirrors the structure of the source Questionnaire and links the defined

[2] See book by Richardson and Ruby, *RESTful web services*, 2008 [23]

questions to the actual data gathered at a point in time.

Bundle: A container resource that allows multiple resources to be grouped together in a single exchange. They are useful for transmitting a complete record (e.g. patient + questionnaire + questionnaire response) in a single transaction.

1.4.4 The FHIR2 module in OpenMRS

The OpenMRS FHIR2[3] module acts as a translation layer between the FHIR standard and OpenMRS' internal data model. It maps elements of the OpenMRS data model to FHIR resources and provides a FHIR REST API.

However, as this module focuses strictly on bridging OpenMRS' internal data structures with established FHIR resources, structures like Quesitonnaire definitions and QuestionnaireResponse records (that do not find clear equivalent in OpenMRS) fall outside the module's mapping scope.

[3] `https://github.com/openmrs/openmrs-module-fhir2`

Chapter 2

Design and Architecture

With the problem landscape established, this chapter addresses the question of how to solve it. The module developed in this master's thesis must satisfy a specific set of constraints: it must integrate natively into OpenMRS without modifying its core, operate in resource-constrained environments, and expose a FHIR interface. All without replicating the infrastructure burden of existing standalone EDC systems. Meeting these constraints requires important architectural decisions. This chapter begins by formalising the requirements that guided those choices, then proposes a data model and system architecture, a database design, a user interface, and finally an interoperability API. Each decision is presented with the reasoning that motivated it, so that the implementation in the next chapter can be understood not only as code, but also as the expression of an architectural vision.

2.1 Requirements Analysis

For this module, requirements were formulated by examining literature related to clinical data collection systems (notably REDCap [24], [25]), the official OpenMRS documentation, and the HL7 FHIR R4 specification[1]. These requirements were separated into two categories: *functional requirements* and *non-functional requirements*.

2.1.1 Functional Requirements

The functional requirements describe all the operations that the module must make available to its users. They were organised into four main categories: questionnaire management, response collection, response export and interoperability API.

[1] `https://hl7.org/fhir/R4/`

Questionnaire Management

- Create a questionnaire directly from the OpenMRS graphical administration interface by providing metadata (title, description, version) and items (questions) of various types (e.g. free text, multiple choice, numeric value, date...).
- Export questionnaires stored in the system as downloadable JSON files.
- Import an existing questionnaire in JSON format compliant with the FHIR Questionnaire R4 specification[2], thereby enabling the reuse of forms generated by other instances or compatible tools.
- Read, edit, and delete questionnaires available in the system.

Response Collection (and management)

- Provide a list of available questionnaires in the system directly from a given patient's medical record.
- When a questionnaire is selected from the list, it is displayed as an interactive form
- Store a FHIR R4 QuestionnaireResponse resource[3] extracted from the filled form.
- Allow to view, edit and delete responses recorded for a specific patient.

Response export

The module should provide tools or a framework enabling collected data to be exported into various formats (e.g. JSON, CSV, etc.). This export tool should be customisable, allow for exports that may or may not contain data identifying the patients, and enable the automatic calculation of new variables derived from the responses.

Given that the module is intended for use in a multi-centre clinical research setting, this complex export tool, like the JSON-formatted questionnaires, must be easily replicable and transferable so that it can be used by all centres participating in the same study.

[2] `https://hl7.org/fhir/R4/questionnaire.html`
[3] `https://hl7.org/fhir/R4/questionnaireresponse.html`

Interoperability API

In order to ensure interoperability, the module should expose an API that supports Create, Read, Update, Delete (CRUD) operations on *Patient*, *Questionnaire* and *QuestionnaireResponse* FHIR resources via standardised REST endpoints. This added interoperability layer must therefore enable the programmatic import and export of questionnaires and responses, in order to facilitate integration with other existing systems.

2.1.2 Non-Functional Requirements

Beyond the features described above, a number of quality requirements have guided architectural decisions:

- **Interoperability**. The module must feature an interoperability layer that allows compatibility with other FHIR R4 third-party systems.
- **Native integration with OpenMRS**. The module must not require any modifications to the OpenMRS core. It must adhere to the conventions of the OpenMRS module framework (use of the OpenMRS API, Spring services and the Hibernate layer) and integrate with existing administration pages and patient records.
- **Open source and deployability**. The module will be released under an open-source licence, and must be easily deployable in resource-constrained settings.
- **Usability**. The module's user interface and export tools should ensure an intuitive and efficient experience.
- **Maintainability and Extensibility**. The architecture must allow to easily add support for new FHIR resources or new API endpoints if necessary.

2.1.3 Use cases

For this OpenMRS module, two main types of users were identified: the "research coordinator", who is responsible for managing questionnaires and data export. And the "clinician" (or any person responsible for data entry at local institutions), who interacts with the module as part of a patient's care journey or study participation. An external system can also interact with the module as a third party using the interoperability API.

The table below lists identified use cases of the module along with the corresponding actor.

Actor	**Use Case**	**Description**
Research Manager	Create a questionnaire	Defines the questionnaire structure (items, types, constraints) via the UI
Research Manager	Import a questionnaire	Uploads a FHIR Questionnaire JSON file
Research Manager	Modify / delete a questionnaire	Updates or removes a questionnaire from the system
Research Manager	Export responses	Retrieves a collection of responses to a given questionnaire of for a given patient in a chosen format
Clinician	Record a response	From the patient record, selects a questionnaire, fills in the items, and saves the response
Clinician	View responses	Visualises previously recorded responses for a patient
Clinician	Modify a response	Corrects an erroneous or incomplete response
External System	Collect questionnaires via API	Queries the interoperability API to retrieve Questionnaire records
External System	Collect responses via API	Queries the interoperability API to retrieve QuestionnaireResponse records

External System	Import a questionnaire via API	Sends a POST request to the interoperability API to create a FHIR Questionnaire record
External System	Import responses via API	Sends a POST request to the interoperability API to add QuestionnaireResponse records for a patient

Table 2.1: Table of identified actors and use cases

2.2 Overall architecture

2.2.1 Integration into the OpenMRS ecosystem

In OpenMRS each module is packaged as an ".omod" (OpenMRS Module) file, which is deployed within the main application's Spring context. A module can extend the OpenMRS database (via Liquibase scripts), expose Spring services (interfaces + implementations), define Spring MVC controllers, and add pages to the user interface.

Based on the requirements described in the previous section, the module developed as part of this master's thesis fits within this model. It consists of two main parts:

- A standard OpenMRS module (Java/Spring backend + HTML and JavaScript frontend) that handles questionnaire management within the administration interface, displays forms in patient records, and stores responses in the OpenMRS data model.
- An integrated HAPI FHIR server that exposes Patient, Questionnaire and QuestionnaireResponse resources via a FHIR R4 REST API.

This dual architecture ensures native OpenMRS compatibility while establishing a FHIR interoperability layer. Consequently, the module's frontend communicates with the backend through a second and dedicated API. The FHIR server is only made available to ensure interoperability.

Architecture diagrams

Figure 2.1 illustrates the main layers of the first part of the system and how they interact. There are five main layers: the presentation layer (OpenMRS UI), the API layer, the service layer (Spring), the data access layer (Hibernate Data Access Objects (DAOs)), and the persistence layer (MySQL database).

On the other hand, figure 2.2 illustrates the main layers of the second part of the system (HAPI FHIR server) and how they interact. There are four main layers. The first one being the entry point layer (HAPI server) with FHIR providers. It is used to handle communication with third-party systems as an interoperability layer. The three other layers are shared with the first part of the system (service layer, data access layer and persistence layer).

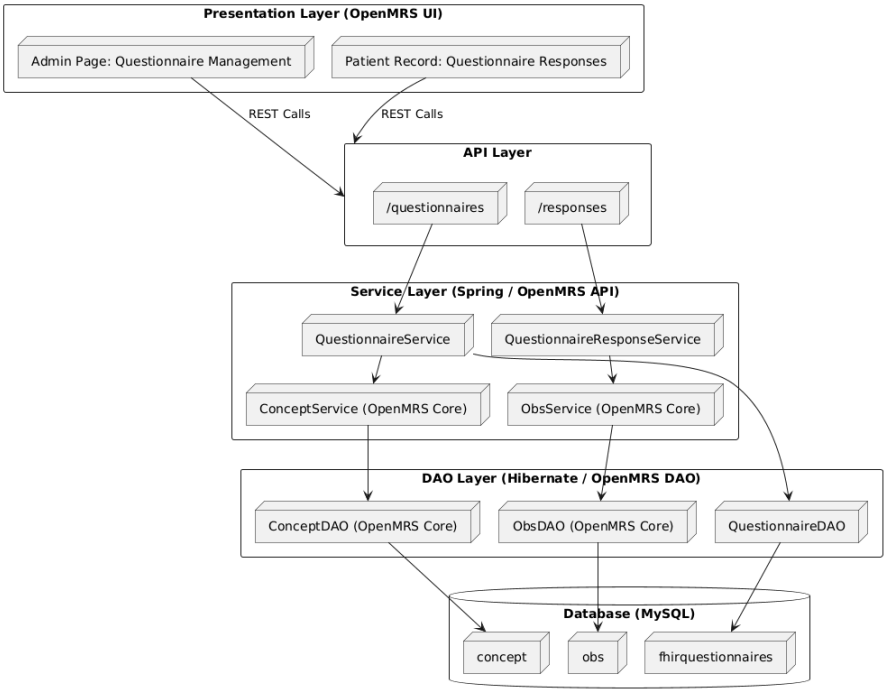


Figure 2.1: Layered architecture of the module

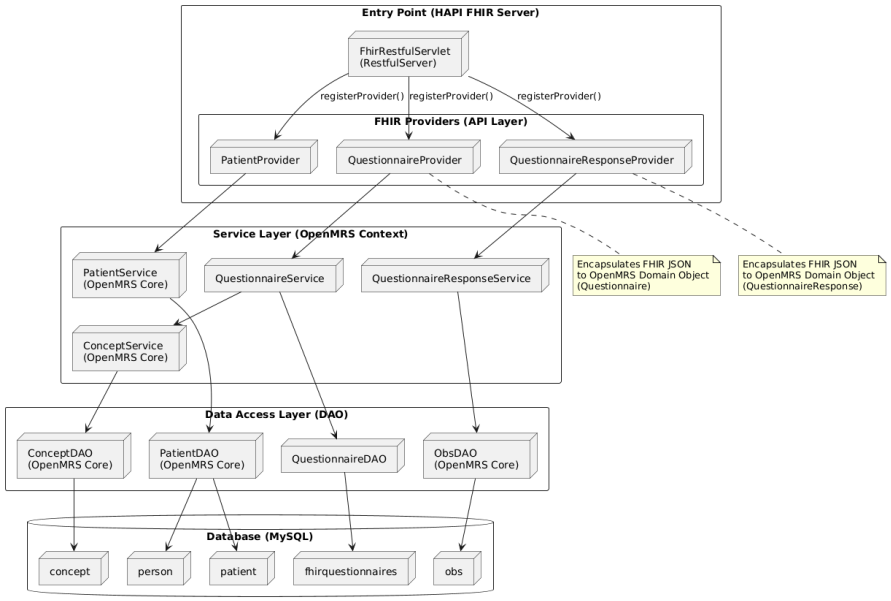


Figure 2.2: Layered architecture of the HAPI FHIR server

2.2.2 Storing FHIR resources in the OpenMRS data model

One of the key design choices needed to build this module was to determine how to store FHIR resources (which are based on a flexible, hierarchical JSON schema) within a relational data model designed for medical records. For this purpose, two main strategies were considered.

The first strategy consisted in separating every FHIR Questionnaire and QuestionnaireResponse's item into distinct SQL records. While this solution enables easier querying, it also creates an unnecessary architectural burden. Indeed, FHIR Questionnaire and QuestionnaireResponse resources are not flat lists. They are complex, recursive structures containing nested groups, activation rules, and extensive metadata. Modeling this depth relationally would require an overly intricate schema prone to breakage whenever the FHIR specification evolves.

In contrast, treating the entire resource as an atomic JSON element allows it to be stored as a single, self-contained record. This approach offers several advantages:

- It fully preserves the resource's structure, including extensions and uncommon elements.
- It simplifies mapping: reading or writing a resource is simply a matter of deserialising or serialising a JSON object.

- It offers flexibility for potential future changes to the specification.

Following this strategy, storing the data is slightly different between questionnaires and responses. Indeed, when it comes to questionnaires, the JSON payload is stored in a dedicated database table (fhirquestionnaires), automatically created by the module.

For responses however, the JSON QuestionnaireResponse is stored in the text field of an OpenMRS observation (inside the "obs" table). This observation is linked to an OpenMRS concept whose name corresponds to the related questionnaire. The observation has class "FHIR Questionnaire" and datatype "Document". This approach allows responses to be anchored within the OpenMRS clinical model (linked to a patient, an encounter, a date), whilst retaining the link to the original questionnaire.

To clarify this model, let's take back the example provided in section 1.3.3. In this example a concept with name "Systolic blood pressure" and "numeric" datatype was created. An "observation" corresponds to the measure of a given patient's systolic blood pressure at a given time. Therefore, inside the database, the "obs" record stores a value (e.g. 104.8) along with the patient identifier and the concept identifier (+ other metadata).

The Fhirquestionnaires module works in a similar way. The questionnaire's definition (in JSON format) is stored in a custom database table and a new concept is created. When a clinician wants to record a response, an observation is created with the JSON QuestionnaireResponse as value, as well as the patient and concept identifiers. In this model, a questionnaire becomes something we want to measure, and a response becomes an observation of that measure, related to a given patient, at a given time. Thus, the semantic link between the questionnaire, the patient and the response is kept inside OpenMRS' specific data model.

2.3 Database design

As briefly introduced previously, when first installed on an OpenMRS instance, the module creates a new table inside the database. This table has the same name as the module. That is, "fhirquestionnaires".

Table 2.2 lists the columns in this database table, as well as their associated SQL type and a brief description.

Column	SQL Type	Null	Description
questionnaire_id	INT (PK, AI)	NO	Auto-incremented internal identifier
uuid	VARCHAR(36)	NO	UUID assigned to the questionnaire
concept	INT	NO	Reference to `concept.concept_id` (Concept assigned to the questionnaire)
name	VARCHAR(255)	NO	Name of the questionnaire, used for display purposes
questionnaire	LONGTEXT	NO	Complete FHIR Questionnaire resource in JSON format
creator	INT	NO	Reference to `users.user_id` (Creator of the record)
date_created	DATETIME	NO	Timestamp when the record was created
changed_by	INT	YES	Reference to `users.user_id` (User who last modified the record)
date_changed	DATETIME	YES	Timestamp when the record was last modified
retired	BOOLEAN	NO	Soft delete flag
date_retired	DATETIME	YES	Timestamp when the record was retired
retired_by	INT	YES	Reference to `users.user_id` (User who retired the record)
retire_reason	VARCHAR(255)	YES	Reason for retiring the record

Table 2.2: Table created in OpenMRS' database

The concept column solves the problem that no two records may share the same UUID. While the `uuid` column allows identifying the record itself in the OpenMRS system, the concept column links the questionnaire to the distinct concept that will be used to record responses. As the module uses observations to store questionnaire responses, this solution allows retrieving both questionnaires and their responses using a unique identifier (`concept_uuid`).

For this reason, both the module's UI and interoperability API refer to the concept's UUID in order to identify a questionnaire. Leaving the record's own UUID used solely for traceability and internal logic.

Note that the columns creator, date_created, changed_by, date_changed,

retired, date_retired, retired_by, and retire_reason are used by convention. They ensure data traceability by tracking every questionnaire's origin, modification history, and soft-deletion status, which is critical for maintaining the integrity and traceability of medical data.

2.4 User interface design

Based on the requirements analysis carried out in section 2.1 (and more specifically on the identified use cases and functional requirements), the user interface was designed to suit two main user profiles: *research coordinator* and *clinician*. Each user profile has access to its own workflow and set of features in a different area of the OpenMRS UI.

2.4.1 Administration page: questionnaire management

The Fhirquestionnaire module's administration interface is accessible from the advanced administration page of OpenMRS. It serves as the central hub for managing the lifecycle of questionnaires, from creation to retiring, ensuring that only valid forms are available for patient data collection. This page provides a dashboard that lists all existing questionnaires, displaying key metadata such as names and identifiers.

From this central view, research coordinators or local administrators can perform several actions:

- **Upload:** Import pre-existing JSON FHIR Questionnaire resources.
- **Visual Builder:** Launch an interactive tool to build new questionnaires from scratch without manual coding.
- **Maintenance:** Rename existing questionnaires, download their raw JSON definitions (to be shared, edited or backed up), and retire obsolete ones to prevent further use.

Once created or uploaded, new questionnaires are immediately made available for selection in the patient records.

Note that compared to the use cases identified in section 2.1.3, research coordinators cannot export questionnaire responses directly from their dedicated UI. This is because the export feature is provided as a customisable script, outside the module's web interface. It will be discussed further in the next chapter.

2.4.2 Patient record: collection and viewing of responses

Integration into the patient record is the most important aspect from a user experience perspective, as this is where clinical data is collected. The module adds a widget to the patient record. When opened from the widget, the module's main page displays :

- The list of QuestionnaireResponses saved for the patient (table showing the questionnaire name and last modification date of the response).
- A button used to access the list of available questionnaires, from where users can record a new response.

When selecting a saved response, an HTML form rendered from the FHIR resource is presented to the clinician. This form is pre-populated with the existing response data.

And when initiating a new response, the clinician is redirected to a submission page where the questionnaire is also rendered as an HTML form. Once submitted, the client-side application converts the form data into a FHIR JSON and transmits it to the server which validates and stores it as an observation about the corresponding patient.

2.5 Interoperability API design

As suggested during requirements analysis, external systems should be allowed to import and collect both questionnaires and their associated response records. And as FHIR is one of the most popular standards for medical data interoperability, it goes without saying that the interoperability API is based on it.

This API exposes a set of REST endpoints to interact with supported resources (Patient, Questionnaire and QuestionnaireResponse). Table 2.3 details the operations available for each resource.

Method	Endpoint	Description
GET	/fhir/Questionnaire	Lists questionnaires (filterable by title)
GET	/fhir/Questionnaire/{id}	Retrieves a questionnaire by its associated concept UUID
POST	/fhir/Questionnaire	Creates a new questionnaire
PUT	/fhir/Questionnaire/{id}	Updates an existing questionnaire (identified by its associated concept UUID)
DELETE	/fhir/Questionnaire/{id}	Soft-deletes a questionnaire (identified by its associated concept UUID)
GET	/fhir/QuestionnaireResponse	Lists responses (filterable by patient, questionnaire)
GET	/fhir/QuestionnaireResponse/{id}	Retrieves a response by its observation UUID
POST	/fhir/QuestionnaireResponse	Saves a new response
PUT	/fhir/QuestionnaireResponse/{id}	Updates an existing response (identified by its observation UUID)
DELETE	/fhir/QuestionnaireResponse/{id}	Soft-deletes a response (identified by its observation UUID)
GET	/fhir/Patient	Searches for patients (filterable by first and last name, identifier, gender or birthdate)
GET	/fhir/Patient/{id}	Retrieves a patient by their UUID
POST	/fhir/Patient/	Creates a new patient
PUT	/fhir/Patient/{id}	Updates an existing patient (identified by their UUID)
DELETE	/fhir/Patient/{id}	Soft-deletes an existing patient (identified by their UUID)
GET	/fhir/metadata	Provides the server's CapabilityStatement

Table 2.3: Interoperability API Endpoints

Chapter 3

Implementation

This chapter documents how the architectural decisions presented in chapter 2 were translated into a working OpenMRS module, describing the technology stack, the project structure, and the implementation of a few system's key features. Rather than listing every class and method, the chapter focuses on the choices that shaped the final technical product (including those that were reconsidered or abandoned). Particular attention is given to the two components that were deemed the more technically relevant: the backend service layer responsible for persisting FHIR resources within OpenMRS's relational data model, and the frontend rendering pipeline that transforms JSON questionnaire definitions into interactive clinical forms.

3.1 Note on the use of AI

The development of the Fhirquestionnaires module followed an iterative process where the role of AI evolved with the complexity of the system. The architectural design and core logic were conceived manually. AI agents were then integrated into the workflow primarily to enhance productivity.

During the initial prototyping phase, a basic mockup was constructed. It had a minimalistic HTML interface. At this stage, AI was employed to refactor and expand the JavaScript codebase to make the module more intuitive to use. It was also used to integrate OpenMRS' styling guidelines, moving from a black-and-white interface in plain HTML to a module that blends visually into the OpenMRS ecosystem. This included the generation of CSS rules required for custom UI elements and animations.

In the backend development, the use of AI was more targeted. Agents served

primarily as debugging tools to identify logical inconsistencies and resolve errors. Furthermore, they were helpful in automating repetitive coding tasks, such as the generation of getters, setters, and standard data transformation methods. This approach allowed the development focus to remain on the core business logic.

3.2 Technology stack

3.2.1 Programming language and framework

The module was developed in Java, in accordance with the requirements of the OpenMRS O2 platform [26]. OpenMRS relies on the Spring Framework for dependency injection, transaction management and the exposure of REST services. The project is divided into two distinct parts: the *api* module (which contains the business logic, service layer and data access layer), and the *omod* (OpenMRS Module) module, which groups together the web controllers, Groovy Server Pages (GSP) views and front-end resources.

Data persistence is handled by Hibernate Object/Relational Mapping [27], with an XML configuration defined in the `Questionnaire.hbm.xml` file. Liquibase manages database changes through the `liquibase.xml` file that describes all the changesets required to create the module's custom table (see section 2.3). This ensures automatic database setup.

Note that the module does not have a `QuestionnaireResponse.hbm.xml` file, as questionnaire responses are stored in the "obs" table (and therefore managed via "ObsService" that is natively provided by OpenMRS).

3.2.2 HAPI FHIR

As explained in the previous chapter, the interoperability API also relies on the FHIR standard to provide a REST server enabling resources to be exposed to compatible external systems.

This API is based on HAPI[1] FHIR. It is an open-source Java library that serves as the reference implementation of the FHIR standard in Java. The library allows to easily build a FHIR server that automatically handles resource validation, JSON/XML serialisation and deserialisation and the generation of the CapabilityStatement[2].

[1] `https://hapifhir.io`

[2] Metadata resource that describes the capabilities and configuration of the FHIR server.

It was chosen because it ensures out-of-the-box compliance with the FHIR standard. The following is a non-exhaustive list of operations that are handled automatically:

- **Resource structure**: The Questionnaire and QuestionnaireResponse resources are validated by the HAPI FHIR parser during import. Any JSON that does not comply with the FHIR standard's resource structure requirements is rejected.
- **XML conversion**: Even if resources are stored as JSON strings, HAPI FHIR allows XML communication with external systems through automatic conversion between JSON and XML representations during request/response processing.
- **Identifiers**: Each resource is identified by a UUID, even Searchsets generated when performing search operation.
- **HTTP status codes**: Server responses comply with the HTTP status codes specified by FHIR[3].

3.2.3 LHC-Forms

The module uses the LHC-Forms library [28] to handle dynamic rendering of FHIR Questionnaire and QuestionnaireResponse resources into interactive HTML forms and the generation of QuestionnaireResponse JSON payloads from HTML form data.

The initial strategy consisted in building a custom JavaScript FHIR engine from scratch. It would parse questionnaires and render forms from them. However, this approach was quickly abandoned during the prototyping phase. Developing a new engine proved to be highly complex, primarily because of the number of edge cases that must be taken into account in a fully FHIR-compliant system. A custom implementation developed alone and in such short time would have inevitably resulted in a less reliable product, prone to bugs in logic handling. It would have failed to support all "small" but important FHIR features without months of additional development.

By integrating LHC-Forms, the module benefits from a mature, community-approved engine that strictly respects the FHIR specification. Furthermore, it

[3] `https://hl7.org/fhir/R4/http.html`

ensures that the generated QuestionnaireResponse resources are structurally valid by default. This approach aligns with the project's goal of delivering a stable, interoperable solution by relying on established standards rather than reinventing complex infrastructure.

3.2.4 Tools and environment

The development was carried out using the OpenMRS SDK, which automates the creation of the project structure and facilitates local deployment. More information about the creation of a new OpenMRS Reference Application Module using the SDK can be found in appendix A. The module was tested on OpenMRS Platform 2.5.9 and MySQL 8.0.

Fhirquestionnaire's front-end relies on GSP views, enhanced with vanilla JavaScript and API calls to REST endpoints designed specifically for the front-end.

3.3 Project structure and package organisation

The project's root contains a parent pom.xml file that declares the two sub-modules, *api* and *omod*. This separation is fundamental: it ensures that the business logic (api) is completely independent of the presentation layer (omod), and that the api module could be imported as a dependency by other modules.

3.3.1 api module

The `org.openmrs.module.fhirquestionnaires.api` package is organised as follows :

- **api/model/**: contains the "Questionnaire" persistence entity. It also includes the "QuestionnaireResponse" class, which does not use a custom table or direct Hibernate mapping as data is persisted as OpenMRS observations in the "obs" table.
- **api/dao/**: defines the Data Access Object "QuestionnaireDao", which serves as the abstraction layer for database interactions. This layer encapsulates all Hibernate criteria queries and session operations required to retrieve, save, or delete Questionnaire entities.
- **api/impl/**: contains the implementations of the services (QuestionnaireServiceImpl, QuestionnaireResponseServiceImpl). This is where most of the business logic happens.

- **provider/**: contains the HAPI FHIR Resource Providers (QuestionnaireProvider, QuestionnaireResponseProvider, PatientProvider). These classes translate incoming FHIR REST requests (CRUD) into internal service calls.
- **web/controller/**: exposes the module's internal REST endpoints (not the interoperability API but custom endpoints used by the module's front-end).

3.3.2 omod module

The `org.openmrs.module.fhirquestionnaires.omod` package organises the user interface according to the OpenMRS pages/fragments model:

- **page/controller/**: contains full-page controllers (patientViewController, RecordResponsePageController). Each controller injects data into the Groovy model exposed to the corresponding GSP view.
- **fragment/controller/**: contains the fragment controller (QuestionnaireResponseListFragmentController) that manages the server-side logic for embedding UI components into the OpenMRS user interface. This controller retrieves patient-specific questionnaire response data and prepares the model data required to render lists of answered questionnaires directly within the patient's medical record view.
- **webapp/pages/**: contains GSP views corresponding to the main pages (patientView.gsp, recordResponse.gsp). They contain the HTML code of the module's UI and include OpenMRS UI elements like headers, footers and basic patient information.
- **webapp/fragments/**: contains the GSP view questionnaireResponseList.gsp that is primarily used to add a clickable button inside the patient's medical record to redirect to the response collection web interface.
- **web/servlet/**: contains FhirRestfulServlet, which initialises and registers the HAPI FHIR server as a servlet accessible at

  "*/openmrs/moduleServlet/fhirquestionnaires/fhir/*"
- **webapp/resources/scripts/**: contains JavaScript files that handle client-side interaction.
- **webapp/resources/css/**: contains CSS files that define web interface styling.

3.3.3 Object Relational Mapping

To bridge the gap between the relational database schema defined in the previous section and the object-oriented nature of OpenMRS, an Object Relational Mapping strategy was designed. The core of this implementation relies on two Java classes: Questionnaire and QuestionnaireResponse. These entities serve as the domain model, and represent the state of FHIR resources while still maintaining a correspondence with the database tables.

The Questionnaire class works as a self-contained entity. It stores the atomic FHIR JSON resource as a string within a private field. Alongside this payload, concept identifiers and metadata (such as the questionnaire's name) are also stored in private fields. This architecture restricts all interactions with the data to public getters and setters, which prevents direct modification of the internal state of the object and ensures data integrity.

On the other hand, the QuestionnaireResponse class operates as a bridge. It does not store the FHIR payload directly within its own fields. Instead, it aggregates data by referencing related entities: Observation, Patient, and Encounter. The class exposes getters and setters that use these relationships to manipulate the FHIR resources stored within the Observation objects, and access metadata stored in the corresponding Patient and Encounter entities.

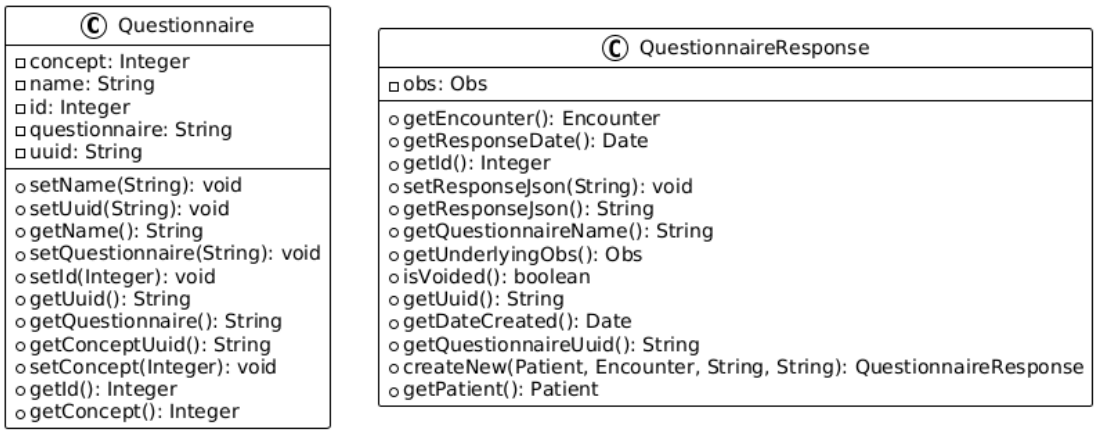


Figure 3.1: UML class diagrams for Questionnaire and QuestionnaireResponse

3.4 Implementation of key features

3.4.1 Backend

Saving a questionnaire

A questionnaire can be added to the system via the interactive questionnaire builder provided by the UI or programmatically via the interoperability API. In both cases, the business logic is handled by the same service layer.

As shown in the code snippet below, when QuestionnaireService receives a questionnaire, it first performs syntax validation using HAPI FHIR to parse the received JSON payload. This ensures that the resource's structure is valid according to the FHIR R4 specification.

```
parser.parseResource(org.hl7.fhir.r4.model.Questionnaire.class,jsonPay
↪  load);
```

If successful, the service carries out the following operations:

- Generating a UUID and inserting it into the "identifier" field of the FHIR JSON payload, ensuring the traceability of the resource and consistency across multiple research centres. Note that if the JSON payload already contains a valid UUID (in the case of an imported questionnaire), it is extracted.
- Creating a new OpenMRS concept of the "FHIR Questionnaire" class, using the OpenMRS ConceptService. This concept is given the questionnaire's name and the generated UUID (or the extracted one if it is not already in use inside the running OpenMRS instance) using the code below.

```
concept.setUuid(uuid);
concept.addName(new ConceptName(questionnaire.getName(),
↪  Locale.ENGLISH));
concept.setDatatype(conceptService.getConceptDatatypeByName("Docu
↪  ment"));
concept.setConceptClass(conceptService.getConceptClassByName("FHI
↪  R
↪  Questionnaire"););
```

- Storing the raw JSON payload with associated metadata in the "fhirquestionnaires" table by passing the questionnaire object to QuestionnaireDao.

Recording patient's response

Similarly to the questionnaire resources, responses can be received from the module's UI (generated by LHC-Forms) or programmatically from an external system through the interoperability API. On the server side, QuestionnaireResponseService handles the following processing:

- Checking JSON syntax using HAPI FHIR.
- Retrieval of the current patient using OpenMRS' PatientService.
- Creating a new Encounter of type "FHIR Questionnaire Response" for the retrieved patient.
- Retrieval of the corresponding concept using OpenMRS' ConceptService (found via the provided UUID).
- Creation and persistence of a new observation inside the encounter, where the concept corresponds to the questionnaire and the value is the JSON payload. This operation is detailed in the following code snippet.

```
1 Obs obs = new Obs();
2 obs.setPerson(patient);
3 obs.setEncounter(encounter);
4 obs.setConcept(conceptService.getConceptByUuid(questionnaireUuid)
  ↪ );
5 obs.setValueText(jsonPayload);
6 obs.setObsDatetime(new Date());
7
8 ObsService obsService = Context.getObsService();
9 obsService.saveObs(obs, "new FHIR Questionnaire Response");
```

3.4.2 Frontend

Creating a new questionnaire

While the LHC-Forms library handles the rendering of existing questionnaires, the creation of new questionnaires is handled by a custom JavaScript Questionnaire Manager.

When the questionnaire management webpage is loaded, the `loadQuestionnaires` function initialises by fetching the list of available questionnaires from the REST endpoint "*/ws/rest/v1/fhirquestionnaires/questionnaires*".

The returned dataset is parsed and rendered into a table, displaying metadata such as the questionnaire's name and UUID. From this table, users can rename, retire, or download the raw JSON payload of any questionnaire.

In order to import existing questionnaires, the upload mechanism leverages the `FileReader` API to parse the provided JSON file on the client side. The system attempts to extract a UUID from the `identifier` array. If a valid UUID is detected, it is preserved to maintain resource consistency. Otherwise, the user is prompted to either let the backend auto-generate one or manually input a valid UUID that will be used to identify the questionnaire. As shown in the following code snippet, the final payload is submitted via a `POST` request to the server.

```
var body = { name: title, questionnaire: questionnaireJson };
if (finalUuid) body.uuid = finalUuid;

fetch('../ws/rest/v1/fhirquestionnaires/questionnaires', {
        method: 'POST',
        body: JSON.stringify(body)
});
```

Alternatively, the interactive form builder allows creating new questionnaires from scratch by maintaining a `builderItems` array that tracks all new questionnaire elements. When the builder modal is opened via `openBuilderModal()`, users can add items using `addBuilderItem()`, which creates objects with properties like linkId, type, text...

Each item is rendered as a card via the `buildItemCardHTML()` function. A card contains fields for question text and answer type selection (string, integer, choice, group, etc.). The UI dynamically adapts thanks to `updateItemTypeUI()` to show answer options for "choice" types or child item containers for "group" types. This allows nested questionnaire structures. Users can also add conditional display rules thanks to the `addEnableWhen()` function to control when items should appear based on other answers.

When the user wants to submit the questionnaire, `buildFhirItem()` is called to recursively convert the `builderItems` array into FHIR Questionnaire JSON format. This payload is then sent to the server using a POST request for server-side storage.

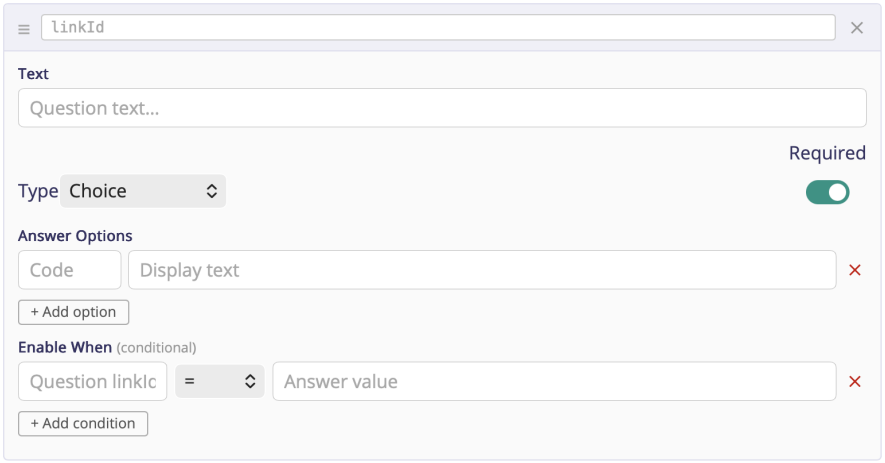


Figure 3.2: Screenshot of an HTML card allowing to define a FHIR Questionnaire item

It should be noted that the purpose of the Fhirquestionnaire module is to design questionnaires that are compatible with the FHIR R4 standard. However, it does not implement all its features. For example, so far, the graphical interface does not allow specifying a version number, a copyright notice or even the maximum length of a response to a given item. All these features can, of course, be added directly by manually editing the JSON payload. But the graphical interface is primarily designed to allow clinicians or researchers without computer science skills to quickly learn to use the tool and create simple yet powerful questionnaires.

Collecting responses

As explained earlier, the rendering of the questionnaire (from a JSON payload to an HTML form) and the collection of responses are handled by the LHC-Forms library.

The `loadQuestionnaire()` function retrieves the specific Questionnaire JSON definition via the REST endpoint "*/ws/rest/v1/fhirquestionnaires/questionnaires/{uuid}*". Upon retrieval, the payload is parsed and passed to `renderLHCForm()`.

The function initialises the LForms context with the patient reference and invokes `addFormToPage()`. LHC-Forms' engine generates the complete form interface, including support for nested groups, conditional display logic, and complex data types.

Data submission again leverages the capabilities of the library. The `submitResponse()`

function calls `LForms.Util.getFormFHIRData()`, which compiles the user's input to the HTML form into a FHIR QuestionnaireResponse JSON payload as shown in the code below.

```
var formDataSource = 'lforms-target'; // form div

var qrData = LForms.Util.getFormFHIRData(
  'QuestionnaireResponse',
  'R4',
  formDataSource
);
```

The resulting JSON is then posted to the response endpoint.

Making modifications to questionnaire responses is done by providing both the QuestionnaireResponse and the original Questionnaire to the LHC-Forms library which renders a pre-populated HTML form. In this case, similarly to new questionnaire responses, the JSON payload is sent to the server. But this time via a PUT request to update the existing record.

3.4.3 Python export script

A simple and easy to modify Python script is included with the module and can be downloaded via its administration interface.

This script provides a basis to interact with the module's internal REST API (not the one exposed by the HAPI FHIR server). Indeed, the module's internal API exposes an anonymising endpoint designed to filter out metadata, identifiers, and patient-specific data other than questionnaire responses.

The decision to implement data export via a provided Python script rather than a native function within the user interface was driven by the need to balance ease of use with customisability. While the script offers an immediate, "plug-and-play" solution for tabular exports, its biggest advantage lies in its adaptability for specific research workflows.

Researchers can use the script as provided or easily extend its functionality to integrate with other APIs (such as the FHIR2 OpenMRS module) for retrieving supplementary patient data (including demographics and medical history for example). Also, the script serves as a flexible foundation for advanced data processing,

allowing users to automate statistical calculations, derive new metrics, define patient cohorts, or apply custom anonymisation protocols.

Furthermore, it is important to note that using such a script allows researchers to retrieve the information in the format of their choice without needing direct access to the OpenMRS instance used for data collection. Researchers can distribute the export script alongside the FHIR Questionnaire to participating data collection centers. Once data collection is complete, local administrators can execute the script on-site to generate standardised exports and transmit only the resulting dataset to the researcher. This workflow is perfectly suited to the needs of multi-centre clinical research and allows maintaining consistency across multiple collection sites.

Chapter 4

Proof of concept: end-to-end scenario

For a module designed to support real multi-centric clinical research, an important test is whether it can support the full lifecycle of a study (from questionnaire design at a coordinating centre to data collection, export and analysis). This chapter demonstrates that capability through a concrete end-to-end scenario involving two fictional clinical centres and the CDC HRQOL-4 survey. The scenario is designed to demonstrate not only the module's graphical interface, but also its interoperability API (showing that the system works as expected whether accessed by a clinician through a browser or by an external FHIR-compliant system). The chapter ends by examining the data export process, which represents the final link between clinical data collection and analysis.

4.1 CDC HRQOL-4 "Healthy Days Measures"

Before diving into the details, it is essential to discuss the CDC HRQOL-4 "Healthy Days Measure" survey [29].

HRQOL-4 is a questionnaire made of four questions designed to assess Health-Related Quality Of Life (HRQOL). It was originally developed by the U.S. *Department of Health and Human Services' Centers for Disease Control and Prevention* in the 1990s. It has since been completed by millions of adults all over the world [29]–[32].

This survey is used as an example questionnaire in this chapter. It is, in fact, fairly short and can easily be converted into a FHIR Questionnaire resource. It thus serves as a real-life example in the visualisations and JSON payloads provided in the following sections. It should also be noted that this short questionnaire is

highly relevant to the subject of this work, given that it is still regularly used in multi-centric studies to this day [33].
For reference, the four questions of CDC HRQOL-4 can be found in appendix B.

4.2 A research coordinator creates and exports a questionnaire

The researcher at Centre A accesses the OpenMRS module's administration page (accessible via Advanced Administration -> Fhirquestionnaires -> Manage Questionnaires). From this page, they create a FHIR version of the CDC HRQOL-4 survey using the simple questionnaire builder.

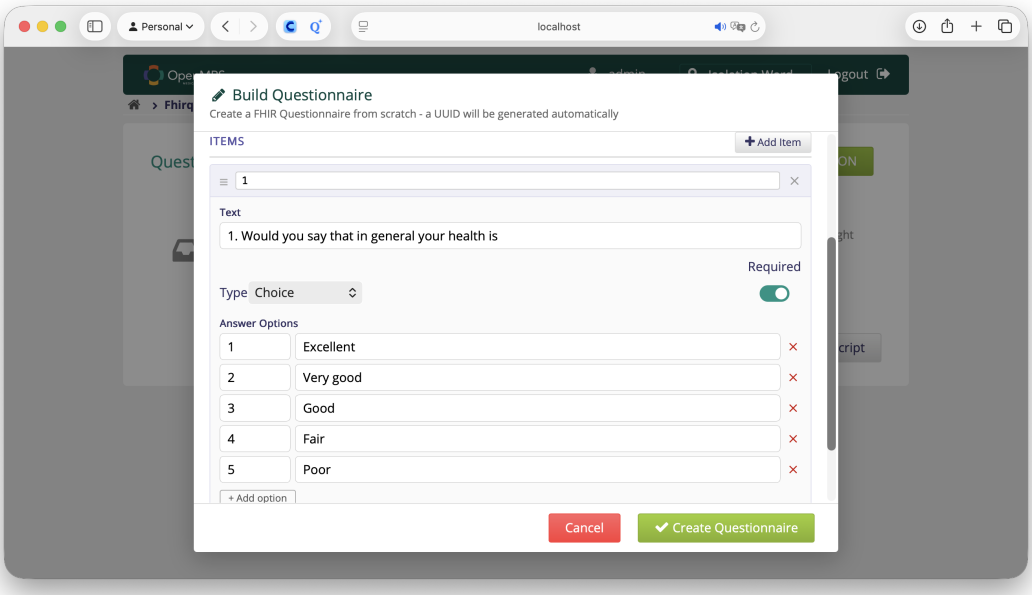


Figure 4.1: Creation of a new questionnaire using the form builder - adding an item (question)

The module's backend then validates the JSON's structure, generates a UUID (for example here, 1c20a0ac-0160-4541-889c-9f3da6d72d87), creates the corresponding OpenMRS concept, and saves the questionnaire in the database. The researcher can now export the questionnaire as a `.json` file by clicking on the download button in the list of available questionnaires. This file can then be sent to other

research sites participating to the study like Centre B.

An example representation of CDC HRQOL-4 survey as a FHIR Questionnaire JSON resource can be found in appendix C. It was generated by the module's form builder.

4.3 Centre B imports the questionnaire

An administrator at Centre B imports the JSON file via the administration page of their OpenMRS instance. Thanks to the presence of the UUID in the imported JSON, Centre B's system creates the questionnaire with the same UUID as that of Centre A. This is crucial for data consistency during cross-centre aggregation: all responses referring to the questionnaire 1c20a0ac-... can be identified as belonging to the same study, regardless of their origin.

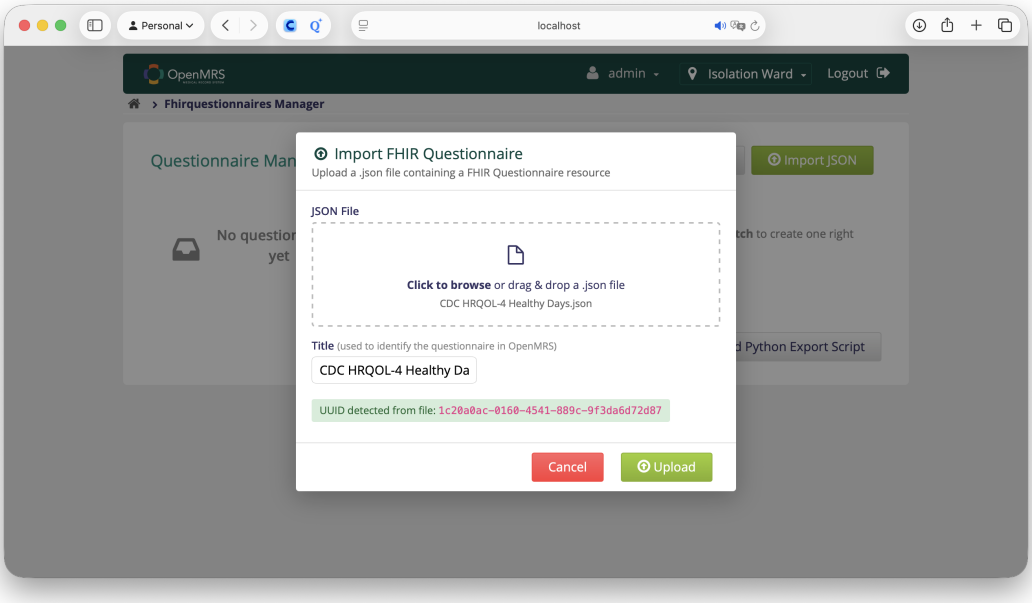


Figure 4.2: Uploading an existing FHIR Questionnaire

Note that if Centre A has access to Centre B's FHIR API, the import can also be carried out programmatically by the research coordinator. Conversely, if Centre B has access to Centre A's FHIR API, the administrator at Centre B can orchestrate the automatic import of the questionnaire.

4.4 A clinician records a response for a patient

A clinician at Centre B is asked to record a QuestionnaireResponse for each patient participating to the study. For this task, two options are possible:

4.4.1 Module's user interface

The clinician opens the medical record of the patient from the OpenMRS user interface. In the Fhirquestionnaires section of the patient record they see a list of responses recorded for the patient. They click on "New Response" to access the list of available questionnaires and select *CDC HRQOL-4 Heathly Days*. The questionnaire is automatically rendered as an HTML form by LHC-Forms.

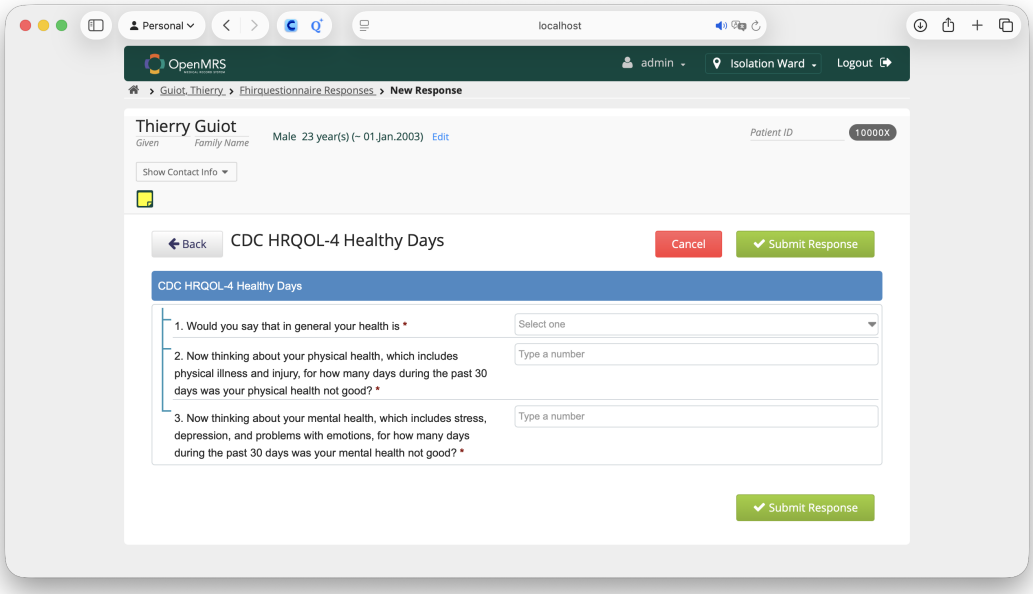


Figure 4.3: Rendering of FHIR Questionnaires in the module's UI

The figure above shows that only 3 of the 4 questions are displayed. This is due to the use of the "enableWhen" constraint (see line 53 in Appendix C). LHC-Forms waits for an answer greater than 0 to be entered in question 2 or 3 before displaying question 4. Even if it is marked as "required", when the display conditions are not met, the user can submit the questionnaire response without answering it. However, once displayed, it becomes mandatory and the system will not allow the response to

be submitted unless an answer is provided for this question. This simple example illustrates the dynamic logic and structural versatility of FHIR resources.

Once the clinician has filled all the required fields, they submit the form. The module then creates an OpenMRS observation with a value corresponding to the QuestionnaireResponse JSON, and confirms the submission with a visual notification in the interface.

4.4.2 Interoperability API

The clinician can also use the FHIR API provided with the module. They can then import the responses programmatically directly in JSON format.

Or, as in the following example, they can use a third-party application compatible with the FHIR Questionnaire resource.

To demonstrate the interoperability of the module developed in this project, *SDC Questionnaire App* was used. It is an application created by the *Lister Hill National Center for Biomedical Communications* [34]. This tool connects to a FHIR server to render questionnaires and generate responses. While this specific application is used as a proof of concept, the underlying FHIR compatibility ensures that any compliant client can interact with the module. This flexibility allows advanced use cases, such as for example the development of dedicated mobile applications for data collection in resource-limited settings with poor connectivity. In such scenarios, a mobile app could store responses locally while offline and automatically synchronise with the OpenMRS server via the FHIR API once network access is restored for example.

Returning to the demonstration example, once opened, the SDC Questionnaire App asks for the base URL of the FHIR server. In the case of the Fhirquestionnaires module, this is the base address of the OpenMRS server, followed by "*/moduleServlet/fhirquestionnaires/fhir/*".

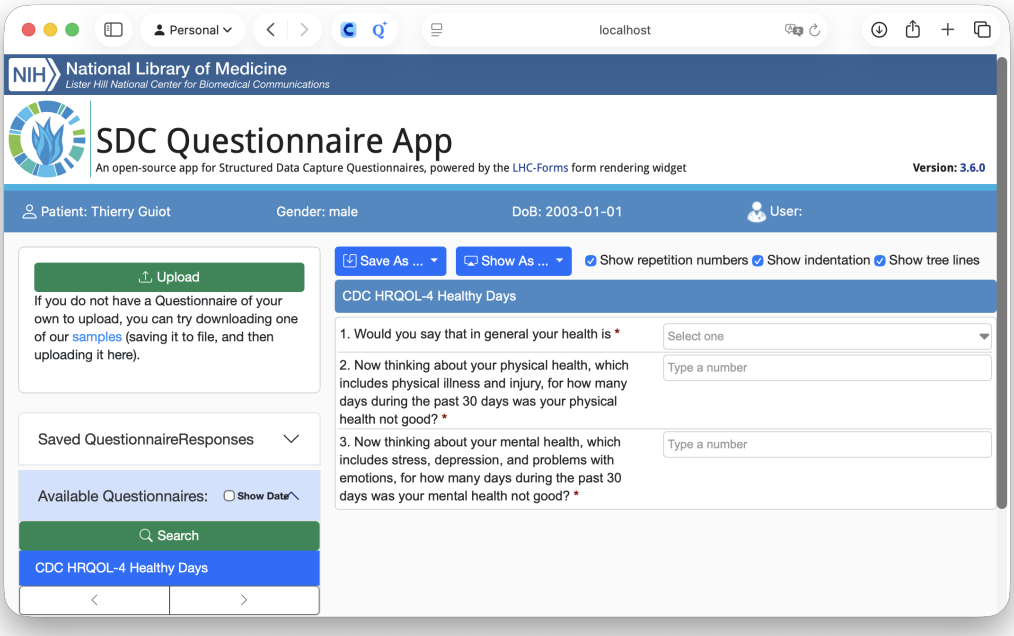


Figure 4.4: Using SDC Questionnaire App (external FHIR-compliant app) to record a QuestionnaireResponse

Once connected to the server, the app asks to select a patient. Once done, SDC Questionnaire App retrieves and displays a list of available questionnaires hosted by the Fhirquestionnaires module in the left-hand navigation panel (as shown in Figure 4.4). When selecting a questionnaire, the user is presented with the form (also rendered via LHC-Forms in this case) to complete the data entry. Once filled, the response is saved inside OpenMRS' data model (exactly like when using the module's own user interface).

4.5 The researcher exports responses for analysis

At the end of the data collection period, the research coordinator wishes to combine responses from all participating centres (here only Centre B). For this, they have two options.

4.5.1 Export via the interoperability API

The research coordinator or local administrator runs a search query on the QuestionnaireResponse endpoint of the HAPI FHIR server, filtered by the questionnaire's UUID. The instance thus returns a FHIR bundle containing all the responses to that questionnaire.

4.5.2 Export via the Python script

As explained in the previous chapter, a Python export script can be downloaded from the module's administration interface (as shown in Figure 4.5 below).

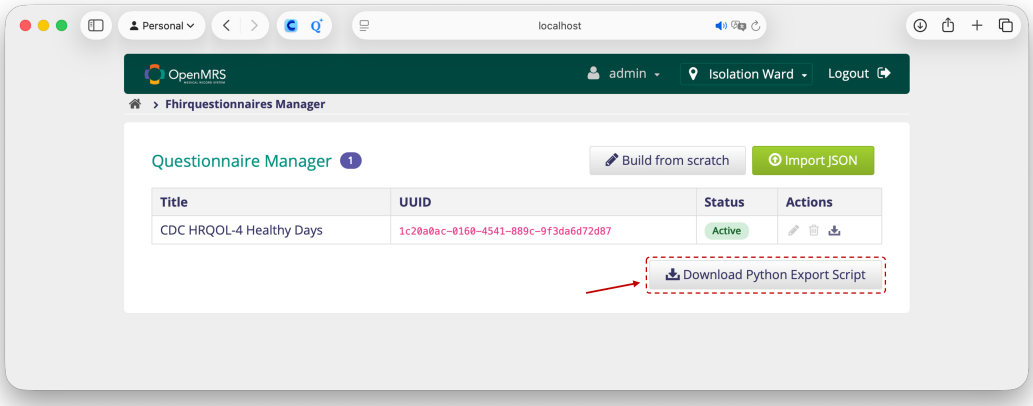


Figure 4.5: Downloading the included export python script

When run, the script first asks for the base URL of the OpenMRS instance (e.g. http://localhost:8000/openmrs/) along with a valid username and password. If the connection to the API is successful, the list of questionnaires hosted by the module is displayed and the user is prompted to select one. They are then asked to choose the export mode (*full* or *anonymised*) and the desired format (*csv* or *xlsx*).

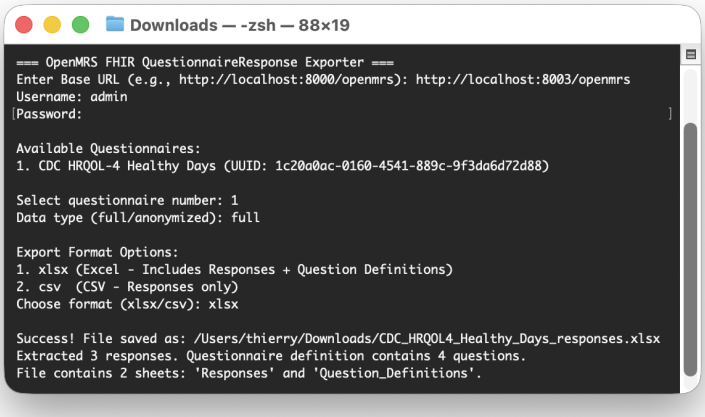


Figure 4.6: Demo of the export python script

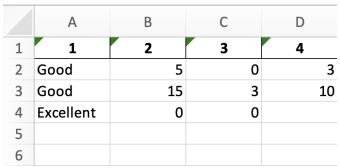
By default, exported data is structured in a tabular format where each row represents a single QuestionnaireResponse record and each column corresponds to a specific FHIR Questionnaire item. The header row identifies each column using the linkId of the associated question.

The "full" data export option augments each record with metadata, including the subject's first and last names, the subject's UUID, the encounter UUID, the response UUID, and the precise timestamp of the observation.

	A	B	C	D	E	F	G	H	I
1	**Subject**	**Subject_UUID**	**Encounter_UUID**	**Response_UUID**	**Obs_Datetime**	**1**	**2**	**3**	**4**
2	Guiot, Thierry	94dc9445-d6f4-4	8221aa52-1fa9-4	bbfed323-6aea-4	2026-05-19 14:54:34.0	Good	5	0	3
3	Guiot, Thierry	94dc9445-d6f4-4	a99c4207-4d7d-4	8b78d3ea-8168-4	2026-05-19 09:51:41.0	Good	15	3	10
4	Guiot, Thierry	94dc9445-d6f4-4	b5cfcf4f-c5e0-4d	1d79467a-0cfe-4	2026-05-14 13:05:58.0	Excellent	0	0	
5									
6									

Figure 4.7: Example of full .xlsx table

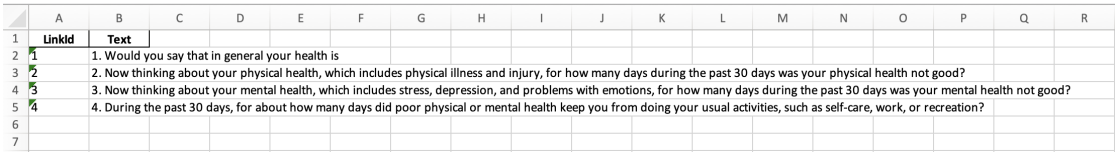
Conversely, when selecting the "anonymized" data export option, the script calls the same API endpoint as for the full export, but with the added "view" query parameter set to "anonymized". The response sent by the module's REST API therefore automatically excludes all patient-specific metadata and identifiers, even in the FHIR QuestionnaireResponse JSON payload.



	A	B	C	D
1	**1**	**2**	**3**	**4**
2	Good	5	0	3
3	Good	15	3	10
4	Excellent	0	0	
5				
6				

Figure 4.8: Example of anonymised .xlsx table

When the data is exported in `.xlsx` format, the resulting file contains two sheets. The second sheet being a table listing the textual description of each item in the related questionnaire.



	A	B
1	**LinkId**	**Text**
2	1	1. Would you say that in general your health is
3	2	2. Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good?
4	3	3. Now thinking about your mental health, which includes stress, depression, and problems with emotions, for how many days during the past 30 days was your mental health not good?
5	4	4. During the past 30 days, for about how many days did poor physical or mental health keep you from doing your usual activities, such as self-care, work, or recreation?
6		
7		

Figure 4.9: Question_Definitions sheet included in the .xlsx file

Conclusion

This master's thesis set out to address a concrete and practically significant gap: the absence of native, standardised research data collection capability within the OpenMRS ecosystem. For institutions in sub-Saharan Africa and Asia (where OpenMRS is most widely deployed, and where multi-centric clinical research is both needed and difficult to conduct) the lack of integrated EDC solutions can seriously undermine multi-centric clinical studies. The central question was whether a FHIR-compliant research data capture module could be built natively into OpenMRS, without modifications to the core system, and without adding infrastructure burdens that resource-constrained sites would have difficulties to sustain.

The answer demonstrated by this work is yes.

The Fhirquestionnaires module integrates into the OpenMRS O2 platform through its module framework. It adds a questionnaire management interface to the OpenMRS administration panel, embeds a response collection widget into the patient record, and exposes a FHIR R4-compliant REST API for interoperability with external systems. Questionnaires are represented as FHIR "Questionnaire" resources and stored as atomic JSON objects in a dedicated database table. Responses are captured as FHIR "QuestionnaireResponse" resources and stored within OpenMRS's clinical data model as observations, preserving the semantic link between a response and the patient it concerns. A Python export script provides a flexible, customisable bridge between the module's internal API and the tabular formats often used for statistical analysis.

The end-to-end scenario in chapter 4 demonstrated that a questionnaire created at one centre can be exported, distributed, and imported at a second centre for data collection. It also demonstrated that the module's FHIR API is interoperable: an unmodified third-party application was connected to the module's interoperability server, retrieved questionnaires, rendered forms, and submitted responses associated to a selected patient.

Key Design Decisions

Two architectural decisions deserve particular reflection.

The first is the choice to store FHIR resources as atomic JSON blobs rather than decomposing them into relational rows. The recursive, extensible structure of the FHIR "Questionnaire" resource would have required a nested relational schema that was both fragile and difficult to evolve. By treating each resource as a self-contained JSON document, the module preserves the full expressiveness of the FHIR standard, simplifies the persistence layer, and makes the system robust to future changes in the FHIR specification. The trade-off that individual response fields cannot be queried directly at the SQL level is acceptable for this use case, where data analysis is performed externally through the export layer rather than through direct database queries.

The second is the decision to use LHC-Forms for questionnaire rendering rather than building a custom FHIR form engine. An attempt was made early in the project to implement a custom JavaScript engine, and it was abandoned when the complexity of full FHIR compliance became clear. The decision to use LHC-Forms was the right one: it provided a mature engine that supports the FHIR R4 questionnaire specification out of the box, and allowed development effort to be redirected to the parts of the system where architectural contribution was required.

Limitations

This work has several limitations that should be acknowledged.

Validation scope

The proof-of-concept provided in chapter 4 demonstrates module's functionalities in a controlled local environment. It does not constitute a deployment in a real clinical setting, and makes no claims about how the module would perform under such conditions. A real-world pilot deployment, ideally in collaboration with an institution using OpenMRS, remains necessary before the module could be considered ready for production use.

User evaluation

No usability study was conducted with representatives of either of the module's two primary user groups (research coordinators and clinicians). The interface was designed with usability principles in mind, but whether it actually meets the needs

of non-technical users in practice is still an open question. In particular, the export mechanism (a Python script run from the command line) may present a barrier for researchers without any programming experience.

FHIR compliance coverage

The module implements the FHIR R4 "Questionnaire" and "QuestionnaireResponse" resources, but does not support the full specification. Several features like version management and copyright fields are not available through the graphical interface. For many use cases this is acceptable, but should still be documented clearly.

Lack of privilege-based access control

OpenMRS uses an advanced permissions system based on actions and roles, in which each action is associated with a privilege. A privilege could be "Add Questionnaire" or "Record QuestionnaireResponse" for example. And a role is a collection of privileges.

The module does not implement a privileges. Currently, there are no authorisation rules that restrict which users can perform specific actions or access particular data within the module. This means that any authenticated user with access to the OpenMRS instance can potentially use all module functionalities, regardless of their intended role (e.g. research coordinator versus clinician). As a result, the security model is far from optimal for a real clinical research environment. This limitation should be addressed before any production deployment to ensure compliance with healthcare data protection requirements.

Single-developer implementation

The module was developed by a single person over the course of a master's thesis. While the codebase is structured according to OpenMRS conventions and is fully open-source, it has not undergone community code review, security audit, or extended testing. These are important prerequisites that should be addressed before any production deployment.

Future Work

The most important immediate next step would be a pilot deployment in a real OpenMRS-using institution, ideally one actively conducting multi-centric research. This would provide evidence about usability and performance under realistic conditions. It would also generate feedback from real users that could be useful to revise

both the interface and export script.

A second priority could be porting the module to OpenMRS O3. This would prove important for the module's long-term relevance.

Closing Reflection

The gap this project addresses is not primarily a technical one. The FHIR standard exists, OpenMRS exists, and the open-source libraries needed to connect them exist. What was missing was a module that assemble them into a deployable and practically usable EDC module. This master's thesis has done that, in a form that can be built upon, deployed, and extended by others.

Patients enrolled in studies deserve the same data quality standards all over the world. Tools like OpenMRS and Fhirquestionnaires are part of what a more equitable global research infrastructure could look like.

Bibliography

[1] X. Zhang, N. Wang, L. Zhang, C. W. Cheng, Z. Liu, F. Liang, W. Xiong, J. Deng, D. Shi, W. Cui, *et al.*, "Reporting assessment of multicenter clinical trial protocols: A cross-sectional study.," *Journal of Evidence-Based Medicine*, vol. 16, no. 1, 2023.

[2] X. Zhang, C. Dong, N. Wang, C. Chan, C. T. Lau, J. Wang, J. Miao, C. Yao, Y. Li, A. Lyu, *et al.*, "Protocol of the consort and spirit extension for multicenter clinical trials," *Frontiers in Public Health*, vol. 11, p. 1 241 152, 2023.

[3] L. Wynants, D. M. Kent, D. Timmerman, C. Lundquist, and B. Van Calster, "Untapped potential of multicenter studies: A review of cardiovascular risk prediction models revealed inappropriate analyses and wide variation in reporting," *Diagnostic and prognostic research*, vol. 3, no. 1, p. 6, 2019.

[4] K. C. Chung, J. W. Song, W. S. Group, *et al.*, "A guide to organizing a multicenter clinical trial," *Plastic and reconstructive surgery*, vol. 126, no. 2, pp. 515–523, 2010.

[5] O. Dekkers, E. v. Elm, A. Algra, J. Romijn, and J. Vandenbroucke, "How to assess the external validity of therapeutic trials: A conceptual approach," *International journal of epidemiology*, vol. 39, no. 1, pp. 89–94, 2010.

[6] I. Fortier, P. Raina, E. R. Van den Heuvel, L. E. Griffith, C. Craig, M. Saliba, D. Doiron, R. P. Stolk, B. M. Knoppers, V. Ferretti, *et al.*, "Maelstrom research guidelines for rigorous retrospective data harmonization," *International journal of epidemiology*, vol. 46, no. 1, pp. 103–105, 2017.

[7] E. F. Patridge and T. P. Bardyn, "Research electronic data capture (redcap)," *Journal of the Medical Library Association: JMLA*, vol. 106, no. 1, p. 142, 2018.

[8] P. A. Harris, R. Taylor, R. Thielke, J. Payne, N. Gonzalez, and J. G. Conde, "Research electronic data capture (redcap)—a metadata-driven methodology and workflow process for providing translational research informatics support," *Journal of biomedical informatics*, vol. 42, no. 2, pp. 377–381, 2009.

[9] I. A. Maré, B. Kramer, S. Hazelhurst, M. D. Nhlapho, R. Zent, P. A. Harris, and M. Klipin, "Electronic data capture system (redcap) for health care research and training in a resource-constrained environment: Technology adoption case study," *JMIR Medical Informatics*, vol. 10, no. 8, e33402, 2022.

[10] M. Cavelaars, J. Rousseau, C. Parlayan, S. de Ridder, A. Verburg, R. Ross, G. R. Visser, A. Rotte, R. Azevedo, J.-W. Boiten, *et al.*, "Openclinica," *Journal of clinical bioinformatics*, vol. 5, no. Suppl 1, S2, 2015.

[11] D. Haak, C.-E. Page, and T. M. Deserno, "Electronic data capture and dicom data management in multi-center clinical trials," in *Medical Imaging 2016: PACS and Imaging Informatics: Next Generation and Innovations*, SPIE, vol. 9789, 2016, pp. 118–123.

[12] M. M. Ngari, N. Waithira, R. Chilengi, P. Njuguna, T. Lang, and G. Fegan, "Experience of using an open source clinical trials data management software system in kenya," *BMC research notes*, vol. 7, no. 1, p. 845, 2014.

[13] Castor EDC, *Castor edc*, `https://castoredc.com/`, Accessed: 2026-04-12.

[14] J. Vielhauer, U. M. Mahajan, K. Adorjan, C. Benesch, B. Oehrle, G. Beyer, S. Sirtl, A.-L. Johlke, J. Allgeier, A. Pernpruner, *et al.*, "Electronic data capture in resource-limited settings using the lightweight clinical data acquisition and recording system," *Scientific Reports*, vol. 14, no. 1, p. 19 056, 2024.

[15] M. O. Akanbi, A. N. Ocheke, P. A. Agaba, C. A. Daniyam, E. I. Agaba, E. N. Okeke, and C. O. Ukoli, "Use of electronic health records in sub-saharan africa: Progress and challenges," *Journal of Medicine in the Tropics*, vol. 14, no. 1, p. 1, 2012.

[16] H. D. Mugauri, M. Chimsimbe, G. Shambira, S. Shamhu, J. Nyamasve, M. Munyanyi, R. Gongora, S. Zizhou, M. Gabida, and R. Makurumidze, "A decade of designing and implementing electronic health records in sub-saharan africa: A scoping review," *Global Health Action*, vol. 18, no. 1, p. 2 492 913, 2025.

[17] B. Jawhari, D. Ludwick, L. Keenan, D. Zakus, and R. Hayward, "Benefits and challenges of emr implementations in low resource settings: A state-of-the-art review," *BMC medical informatics and decision making*, vol. 16, no. 1, p. 116, 2016.

[18] A. Bashiri and M. Ghazisaeedi, "Open mrs softwares: Effective approaches in management of patients' health information," *Int J Commun Med Publ Health*, vol. 4, no. 11, p. 4, 2017.

[19] OpenMRS, *About us*, `https://openmrs.org/about/`, Accessed: 2026-04-12.

[20] D. Bender and K. Sartipi, "Hl7 fhir: An agile and restful approach to healthcare information exchange," in *Proceedings of the 26th IEEE international symposium on computer-based medical systems*, IEEE, 2013, pp. 326–331.

[21] C. N. Vorisek, M. Lehne, S. A. I. Klopfenstein, P. J. Mayer, A. Bartschke, T. Haese, and S. Thun, "Fast healthcare interoperability resources (fhir) for interoperability in health research: Systematic review," *JMIR medical informatics*, vol. 10, no. 7, e35724, 2022.

[22] M. Ayaz, M. F. Pasha, M. Y. Alzahrani, R. Budiarto, and D. Stiawan, "The fast health interoperability resources (fhir) standard: Systematic literature review of implementations, applications, challenges and opportunities," *JMIR medical informatics*, vol. 9, no. 7, e21929, 2021.

[23] L. Richardson and S. Ruby, *RESTful web services*. " O'Reilly Media, Inc.", 2008.

[24] P. A. Harris, R. Taylor, B. L. Minor, V. Elliott, M. Fernandez, L. O'Neal, L. McLeod, G. Delacqua, F. Delacqua, J. Kirby, *et al.*, "The redcap consortium: Building an international community of software platform partners," *Journal of biomedical informatics*, vol. 95, p. 103 208, 2019.

[25] G. Nicora, L. Sacchi, V. Tibollo, E. Cigliutti, M. Caretti, F. Falchini, M. C. Mauro, A. Fasano, M. Siotto, A. Pavan, *et al.*, "Digitizing rehabilitation outcomes and assessing data quality in clinical trials: Implementing validated scales in redcap for a stroke rct," *BMC Medical Research Methodology*, 2026.

[26] OpenMRS, *Openmrs developer manual - architecture*, `https://devmanual.openmrs.org/technology/`, Accessed: 2026-05-05.

[27] Hibernate, *Object/relational mapping*, `https://hibernate.org/orm/`, Accessed: 2026-05-05.

[28] Lister Hill National Center for Biomedical Communications, *Lhc-forms*, `https://github.com/lhncbc/lforms`, Accessed: 2026-05-05.

[29] D. G. Moriarty, M. M. Zack, and R. Kobau, "The centers for disease control and prevention's healthy days measures–population tracking of perceived physical and mental health over time," *Health and quality of life outcomes*, vol. 1, no. 1, p. 37, 2003.

[30] T. Mielenz, E. Jackson, S. Currey, R. DeVellis, and L. F. Callahan, "Psychometric properties of the centers for disease control and prevention health-related quality of life (cdc hrqol) items in adults with arthritis," *Health and quality of life outcomes*, vol. 4, no. 1, p. 66, 2006.

[31] U. B. Aslan, U. Cavlak, N. Yagci, and E. Baskan, "Reliability and validity of the turkish version of the cdc hrqol-4 scale in patients with chronic low back pain.," *Pakistan Journal of Medical Sciences*, vol. 26, no. 4, 2010.

[32] J. Toet, H. Raat, and E. J. van Ameijden, "Validation of the dutch version of the cdc core healthy days measures in a community sample," *Quality of Life Research*, vol. 15, no. 1, pp. 179–184, 2006.

[33] L. Sage, E. Hart, N. Meyer, O. Hnilicka, A. Penkin, T. C. Poteat, R. Aguayo-Romero, B. A. Comstock, S. P. Committee, G. W. Dy, *et al.*, "Support for transgender and nonbinary individuals seeking vaginoplasty (strive) study: Protocol for a national randomised pragmatic trial," *BMJ open*, vol. 16, no. 2, e114287, 2026.

[34] Lister Hill National Center for Biomedical Communications, *Sdc questionnaire app*, `https://github.com/lhncbc/lforms-fhir-app`, Accessed: 2026-05-05.

Appendices

A Creating a New OpenMRS Reference Application Module

This guide details the procedure for creating a new **OpenMRS Reference Application module** from scratch using the OpenMRS SDK. It includes the necessary Maven configurations to ensure compatibility with Java 8 and modern JVMs.

Prerequisites

Before beginning, ensure the following are installed:

- **Java JDK 8**
- **Apache Maven**
- **OpenMRS SDK**

  Verify the installations by running:

```
java -version
mvn -version
```

Step 1: Create a New OpenMRS Project

Execute the OpenMRS SDK project creation command from the terminal:

```
mvn openmrs-sdk:create-project
```

Step 2: Choose Project Type

When prompted with the project type selection:

```
What kind of project would you like to create?:
1. Platform module
2. Reference Application module
3. Open Web App
4. Content Package
```

Select option 2 to create a **Reference Application module**. This type is suitable for UI and backend extensions targeting the OpenMRS Reference Application.

Step 3: Provide Module Metadata

Enter the following details when prompted:

- **Module ID**: e.g., `mycustommodule`
- **Module name**: e.g., `My Custom Module`
- **Module description** (optional): e.g., `Adds custom features to OpenMRS`
- **Initial version** (optional): e.g., `1.0.0-SNAPSHOT`
- **Lowest OpenMRS version to support** (optional): e.g., `2.5.0`

Upon completion, the SDK generates a multi-module Maven project.

Step 4: Update the `omod` Module Maven Configuration

To ensure compatibility with newer JVMs and enforce Java 8 compilation, modify the `omod/pom.xml` file. Insert the following configuration within the `<build><plugins>` section:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>3.2.5</version> <!-- or latest -->
    <configuration>
        <argLine>--add-opens java.base/java.lang=ALL-UNNAMED</argLine>
    </configuration>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
```

```
        <source>8</source>
        <target>8</target>
    </configuration>
</plugin>
```

Rationale:

- The `maven-surefire-plugin` with `-add-opens` prevents reflective access errors on newer JVMs.
- The `maven-compiler-plugin` ensures compilation targets Java 8, as required by OpenMRS.

Step 5: Update the `api` Module Maven Configuration

Apply the same configuration to the `api/pom.xml` file to maintain alignment between modules. Add the following within the `<build>` section:

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>3.2.5</version> <!-- or latest -->
        <configuration>
            <argLine>--add-opens java.base/java.lang=ALL-UNNAMED</argLine>
        </configuration>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
            <source>8</source>
            <target>8</target>
        </configuration>
    </plugin>
</plugins>
```

Step 6: Build the Module

From the project root directory, execute the build command:

```
mvn clean install
```

B CDC HRQOL-4

The CDC HRQOL-4 (Health-Related Quality of Life) is a surveillance instrument developed by the Centers for Disease Control and Prevention to measure population health status. Often referred to as the "Healthy Days Measure", it captures four core dimensions of health: self-rated general health, recent physical health, recent mental health, and activity limitations. These four questions are as follows:

- Would you say that in general your health is; Excellent, Very good, Good, Fair or Poor?
- Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good?
- Now thinking about your mental health, which includes stress, depression, and problems with emotions, for how many days during the past 30 days was your mental health not good?
- During the past 30 days, for about how many days did poor physical or mental health keep you from doing your usual activities, such as self-care, work, or recreation?

C CDC HRQOL-4 as a FHIR Questionnaire

The following is a JSON representation of the CDC HRQOL-4 as a FHIR Questionnaire. It was generated using the interactive questionnaire builder included in the Fhirquestionnaires module.

```
{
  "resourceType": "Questionnaire",
  "title": "CDC HRQOL-4 Healthy Days",
  "status": "active",
  "name": "CDC_HRQOL-4",
  "description": "The standard 4-item set of Healthy Days core
↪  questions used in BRFSS, NHANES, and HOS.",
  "identifier": [
    {
      "value": "1c20a0ac-0160-4541-889c-9f3da6d72d87"
    }
  ],
  "item": [
    {
      "linkId": "1",
      "text": "1. Would you say that in general your health is",
      "type": "choice",
      "required": true,
      "answerOption": [
        {
          "valueCoding": { "code": "1", "display": "Excellent" }
        },
        {
          "valueCoding": { "code": "2", "display": "Very good" }
        },
        {
          "valueCoding": { "code": "3", "display": "Good" }
        },
        {
          "valueCoding": { "code": "4", "display": "Fair" }
        },
        {
          "valueCoding": { "code": "5", "display": "Poor" }
        }
      ]
    },
    {
```

```
      "linkId": "2",
      "text": "2. Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good?",
      "type": "integer",
      "required": true
    },
    {
      "linkId": "3",
      "text": "3. Now thinking about your mental health, which includes stress, depression, and problems with emotions, for how many days during the past 30 days was your mental health not good?",
      "type": "integer",
      "required": true
    },
    {
      "linkId": "4",
      "text": "4. During the past 30 days, for about how many days did poor physical or mental health keep you from doing your usual activities, such as self-care, work, or recreation?",
      "type": "integer",
      "required": true,
      "enableWhen": [
        { "question": "2", "operator": ">", "answerString": "0" },
        { "question": "3", "operator": ">", "answerString": "0" }
      ]
    }
  ]
}
```

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**