High-Level Design Document: Configurable printing of forms

11-12-2025

SolDevelo

Table of contents

Executive Summary

Objective: To implement a scalable, professional printing solution for OpenMRS that allows users to generate generic PDF documents (Discharge Summaries, Referrals, Test Results) based on existing clinical data.

The Solution: We will build a "Zero-Config" Generic Printing Engine by extending the existing Patient Documents Module (an MSF/Madiro/Mekom initiative). Instead of designing manual templates for every form, the system will programmatically read the structure of any clinical form and automatically generate a matching, professionally formatted PDF.

Key Decisions:

- **Strategy:** Rejected external tools to avoid vendor lock-in and high maintenance costs.
- **Technology:** Leverages the Reporting Module for data and Apache FOP for rendering.
- **Effort:** Estimated ~18.5 Man-Days to deliver a fully dynamic engine that supports current and future forms without additional development.

1. Overview

1.1 Feature

This High-Level Design (HLD) document comprehensively outlines the strategic, functional, and architectural approach necessary for introducing a robust, flexible, and configurable printing mechanism within the OpenMRS (MRS) system. The core objective is to empower users to generate customizable, formatted printouts of various forms, records, and summaries based on specific clinical and administrative requirements, ensuring that medical professionals can easily extract and print medical records for external use or record-keeping.

1.2 Definitions

Term	Definition
MRS	Medical Records System (referring to

	OpenMRS in this context).
Clinical Form	A Clinical Form is a highly structured metadata object defined by a JSON Schema, part of OpenMRS O3 feature-set.
Patient Documents Module	The OpenMRS module: https://github.com/openmrs/openmrs-module-patientdocuments. This module currently handles patient-related document generation, possessing key PDF generation capabilities. It was developed during a joint initiative of MSF, Mekom and Madiro.
Reporting Module	The OpenMRS module: https://github.com/openmrs/openmrs-module-reporting. A long-established, core module present in most OpenMRS implementations, that provides foundation for report generation.
XML	Extensible Markup Language. Used as an intermediary data format for report generation.
XSLT	Extensible Stylesheet Language Transformations. Used to transform the raw XML data into the final PDF-ready document structure, enabling configuration.

2. Requirements

The system shall provide a scalable, rule-based printing engine capable of rendering clinical forms and encounters without requiring dedicated configuration for individual document types.

2.1 Core Functional Requirements (The "Generic Engine")

Instead of static templates for specific documents (e.g., Discharge Summaries), the system must dynamically generate printouts based on the underlying data structure of the selected form.

- **R.2.1.1 Dynamic Structure Mapping:** The system must automatically interpret and format form elements according to a universal logic:
    - Section Labels convert to Section Headings;
    - Question Names convert to Field Labels;
    - Answers/Values convert to Body Text (formatting varies by data type, e.g., dates vs. text blocks).
- **R.2.1.2 Zero-Config Scalability:** The printing process must automatically apply to all current and future forms. Adding a new clinical form to the system must not require the creation of a corresponding print template.
- **R.2.1.3 Null Value Handling:** To maintain professional conciseness, the system should allow for global rules regarding empty fields (e.g., "Hide questions with no answers" or "Display as N/A").

2.2. User Interaction

- **R.2.2.1 Selection Granularity**: The user must be able to select the scope of the printout at the following levels:
    - **Single Form:** Print a specific form (e.g., "Admission Note").
    - **Single Encounter:** Print all data associated with a specific visit.
    - **Multi-Encounter (Priority: High):** Aggregate selected forms from multiple encounters into a single chronological PDF.
- **R.2.2.2 Contextual Awareness:** The printout must automatically inherit the context of the selected item (e.g., Patient Name, Visit Date) and display it in the header, as defined in Section 2.3.

2.3 Universal Formatting Rules (The "Style Guide")

The Generic Engine must apply the following style attributes to the mapped elements defined in R.2.1.1. These rules ensure consistency across all generated documents.

Attribute	Specification	Detail
Paper Size	A4 (210 × 297 mm)	Standardized paper size.
Orientation	Portrait	Standard document orientation.
Font Family	Arial	Standard professional font.
Margins	2 cm (all sides)	Ensures adequate white space.
Line Spacing	1.15 – 1.5	Customizable for readability.
Font Sizes & Styles	Document Title	14 pt, Bold, Centered
	Section Headings	12 pt, Bold, Left-aligned
	Body Text	11 pt, Regular, Justified
Header Content	Health Facility name + Document type (e.g., “Discharge Summary”)	Essential identification details.
Footer Content	Page number + confidentiality disclaimer + health facility address	Mandatory legal and contact information.
Emphasis (High Priority)	Patient Identifiers	Name, ID, Date of Birth/Age must be Bold, 12 pt.
	Abnormal Results	Abnormal lab results must be Bold or Highlighted for immediate attention.

	Instructions	Patient instructions (e.g., medications, follow-up schedule) must be Bold or presented in a Larger font (e.g., 12pt).

3. Key Findings

A thorough analysis of potential printing solution approaches was conducted, comparing external reporting engines (e.g., JasperReports), a fully custom build, and leveraging existing OpenMRS modules.

The analysis was heavily influenced by the requirement for a "Generic/Zero-Config" engine. We specifically evaluated whether a template-based approach could scale to meet the volume of clinical forms without requiring a dedicated design file for every single encounter type.

3.1 Integrate External Template Solutions

This approach involves plugging in a third-party reporting engine that uses a WYSIWYG (What You See Is What You Get) editor to design printouts.

Benefits	Risks
Visual Precision: Offers pixel-perfect control over document layout via a drag-and-drop editor.	**The "Template Trap":** This solution requires a 1:1 mapping between a clinical form and a print template. Every new form introduced in the system would require a designer to manually create a new template file.
Feature Rich: Advanced charting and formatting out-of-the-box.	**Not Generic:** It relies on static templates rather than dynamic interpretation of form structures. Scaling this to hundreds of forms would represent a massive, ongoing maintenance cost.
	External Dependency: Introduces a heavy dependency on third-party libraries and a

	separate server/process for rendering.
	Steep Learning Curve: Requires the team to learn a specialized, complex proprietary tool that is distinct from the core OpenMRS tech stack.

3.2 Build a Fully Custom Solution

Building a printing engine from scratch specifically for this implementation.

Benefits	Risks
Perfect Integration: Guarantees the best fit for OpenMRS data structures.	**High Cost:** Extremely high effort and time investment required.
UX Consistency: Familiar experience for users.	**Reinventing the Wheel:** Requires building basic rendering logic that already exists in other tools.

3.3 Extend Existing Modules

Benefits	Risks
Strategic Alignment: Directly builds upon the MSF/Madiro/Mekom joint initiative. This ensures our solution aligns with the client's existing ecosystem and facilitates future collaboration.	**Legacy Constraints:** Potential limitations in how current modules handle complex nesting or specific data types, requiring extension.
Scalability: Best positioned to support the "Generic Engine" requirement.	**Adaptation:** Requires careful adaptation to ensure the generic rendering looks professional across different form types.
Native Data Access: Direct access to OpenMRS metadata (concept names, answers) without external mapping layers.	

3.4 Conclusion

Based on the analysis, we have decided to proceed with **4.3 Extend Existing Modules**.

We rejected 4.1 Integrate External Template Solutions because it fails the critical "Zero-Config" scalability requirement. A template-based approach would have forced the team to design and maintain a specific layout file for every current and future form - a "huge task" that is not sustainable. Furthermore, it would have introduced a steep learning curve for a specialized tool outside the core OpenMRS stack.

By selecting 4.3 Extend Existing Modules, we secure two major advantages:

- **Technical:** We will build a single Generic Rendering Engine that dynamically interprets form metadata, ensuring all forms can be printed without manual configuration.

- **Strategic:** We leverage and enhance the **Patient Documents Module**, validating the investment previously made by the **MSF, Madiro, and Mekom** joint initiative and ensuring the solution remains aligned with the client's existing ecosystem.

4. Technical Specification

The solution is architected around a "Zero-Config" principle. Instead of creating static XSLT templates for every clinical form, we will build a dynamic engine that reads the OpenMRS Form Schema (JSON) to reconstruct the document structure programmatically.

This architecture leverages the Patient Documents Module (an MSF/Madiro/Mekom joint initiative) for its robust XML-to-PDF transformation pipeline (Apache FOP), while extending the Reporting Module to handle complex data mapping.

4.1 Architecture Overview

The process follows a linear transformation pipeline:

1. **Trigger:** User requests a printout for an Encounter.
2. **Structure Retrieval:** System fetches the OpenMRS Form Definition (JSON) to understand the visual hierarchy (Sections, Questions).
3. **Data Retrieval:** System fetches the Encounter Data (Obs).
4. **Dynamic Merging:** The new Renderer maps the flat data (Obs) into the hierarchical Form structure.
5. **Rendering:** A Single Universal XSLT applies styling rules to generate the PDF.

4.2 Core Component: EncounterXmlReportRenderer (New)

This component is the "brain" of the solution. It replaces the manual creation of templates (required in the current Patient Documents module) with code logic.

- **Responsibility:** It functions as a bridge between the raw clinical data and the visual form definition.
- **Logic (The "Schema-Driven" Approach):**
    - Unlike standard renderers that simply dump data, this renderer accepts the Form Template UUID (JSON) as a parameter.
    - It parses the JSON to identify Sections, Pages, and Field Orders.
    - It sorts and groups the retrieved Observations (Obs) to match this structure exactly.
    - Output: It generates a highly structured XML stream that mirrors the user interface rather than a flat list of database rows.

4.3 PDF Generation: Universal XSLT & Apache FOP

To satisfy the "Universal Formatting Standard" (Requirement 2.5) without per-form configuration:

- **Universal XSLT:** We will develop a single, master XSLT file. This template is "content-agnostic" - it does not contain hardcoded references to specific medical forms (e.g., "Diabetes Intake"). Instead, it contains logic to render generic elements:
    - *Rule*: "If <Section> is found, print Header in Bold 12pt."
    - *Rule*: "If <Obs> is found, print Label in Grey and Value in Black."
- **Apache FOP:** The existing FOP engine from the Patient Documents Module will consume this XML/XSLT pair to generate the final PDF.

4.4 Backend Integration (Async Controller Layer)

- **Controller:** EncounterDataPdfExportController.
- **Execution Model:** Due to the potential latency of PDF generation, this controller will not return the PDF immediately.
    - Action: It accepts the request, spawns a background task (leveraging existing OpenMRS Reporting), and returns a unique Id.
    - Status Endpoint: A lightweight endpoint will be exposed to check progress.

4.5 Frontend Integration

To support the requirement for aggregating records (e.g., "Print all notes from this hospitalization"), the frontend interaction will evolve beyond a simple "Download" button into a dedicated Print Context Modal.

- **Entry Point:** A "Print Records" action in the Dashboard.
- **Selection Modal:** Allows users to check multiple historical encounters (e.g., "Include previous Discharge Summary").
- **UX Pattern (Async Handling):**
    - **Initiate:** Frontend sends payload and receives unique Id.
    - **Wait State:** UI displays a non-blocking "Generating Report..." spinner or progress bar, polling the status endpoint.
    - **Completion:** Once the backend reports "COMPLETED," the UI replaces the spinner with a "Download PDF" button or automatically triggers the browser download.

4.6 Performance Considerations

Since report generation is computationally expensive, the system must enforce strict reliability patterns:

- **Asynchronous Processing:** As detailed in 5.4/5.5, the Frontend must never "hang" waiting for a response. The "Request ID to Poll to Download" pattern is mandatory to prevent browser timeouts on large datasets.
- **Timeout Handling:** If the backend process exceeds a defined threshold (e.g., 60 seconds), the task must be terminated gracefully, and the Frontend must display a user-friendly error ("Report generation timed out. Please select fewer encounters.").
- **Empty State Handling:** If a user selects an encounter that contains no data (no Observations), the engine must generate a valid PDF with a "No Data Recorded" placeholder rather than crashing or returning a corrupt file.
- **Concurrency:** The backend implementation should consider a thread pool to limit the number of simultaneous PDF generation tasks, preventing server memory exhaustion during peak hours.

4.6.1 Expected baseline performance

Based on standard Apache FOP benchmarks in OpenMRS environments:

- **Single Form (1-3 pages):** < 3 seconds.
- **Discharge Summary (Composite):** ~5 seconds.
- **Full Patient Record (50+ Encounters, 100+ pages):** 30–90 seconds.

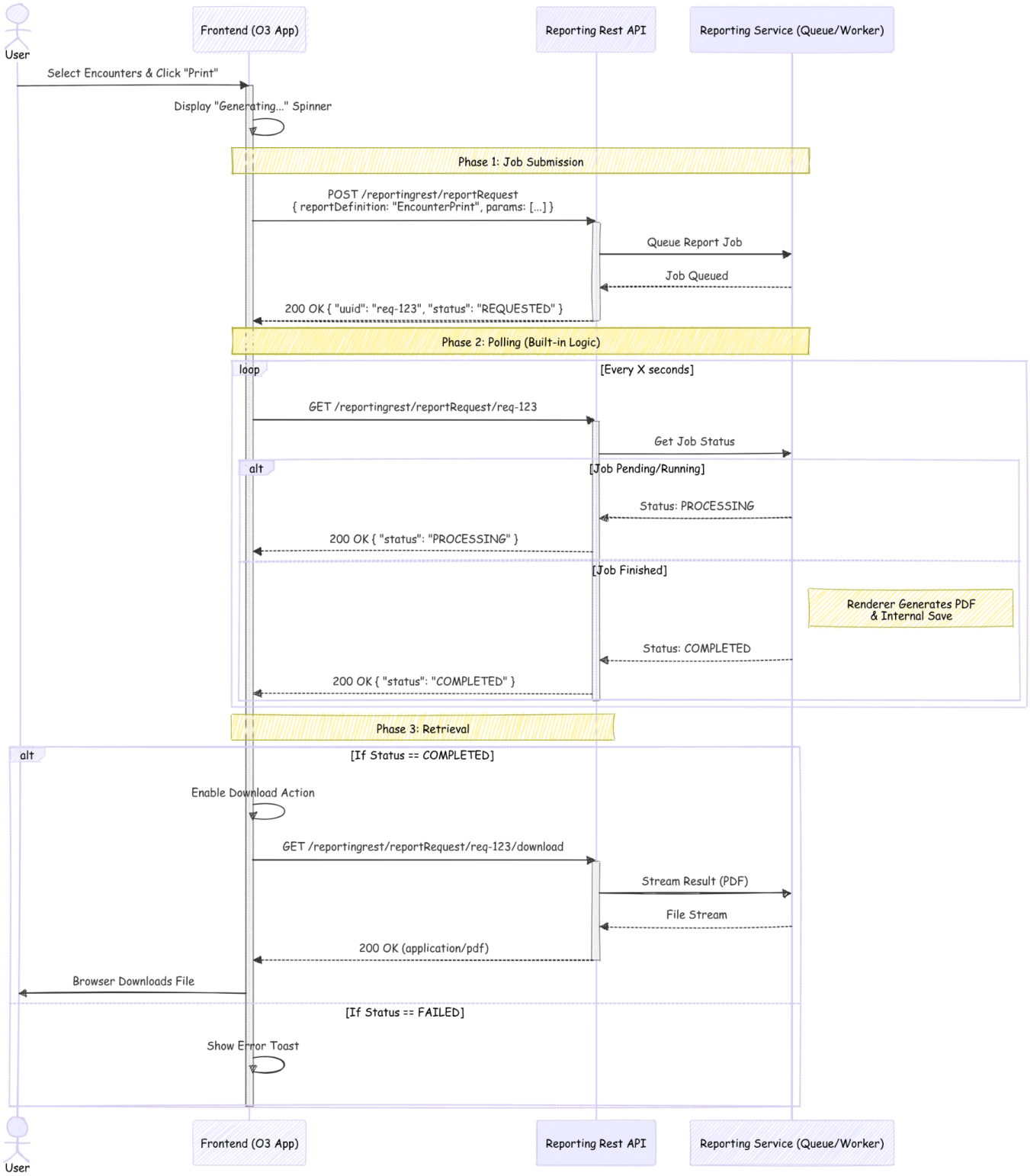
4.7 Error Handling Strategy

Given the asynchronous nature of the "Request to Poll to Download" flow, error handling must be explicitly managed at both the Backend (Job Execution) and Frontend (Status Interpretation) layers.

- **Backend Handling (Reporting Module):**
  - **Graceful Failures:** The EncounterXmlReportRenderer must not crash the server thread. Instead, it must log the stack trace and propagate a clean error message to the ReportRequest object, marking the job status as FAILED.
  - **Empty Data:** If an encounter contains no Observations, the system will not throw an error. It will generate a valid PDF containing the Header/Footer and a placeholder message: "No clinical data recorded for this encounter."
- **Frontend Handling (Polling Loop):**
  - The polling logic in the Frontend will explicitly check for a fail state.
  - Upon detecting a failure, the UI will stop the spinner and display a Toast Notification or Modal Alert containing the specific error message returned by the API (e.g., "Generation Failed: Form Template not found for UUID...").
- **Specific Failure Scenarios:**

Scenario	System Behavior	User Feedback
Missing Form Definition	Backend catches missing Schema UUID. Job marked FAILED.	Error: "Unable to print: The form structure could not be retrieved."
Invalid/Corrupt Data	XML generation fails. Job marked FAILED.	Error: "Data processing error. Please contact support."
Timeout (Slow Generation)	Reporting Module or Load Balancer kills the thread.	Frontend detects HTTP 504 or lack of status update. Error: "Request timed out. Please try fewer encounters."

4.8 Sequence Diagram: Async Polling



5. Effort Estimation

The following estimates cover the design, implementation, and testing of the Generic Schema-Driven Printing Engine (as defined in Section 5).

5.1 Breakdown by Component

Domain	Task Description	Estimate (Man-Days)
Backend	Core Logic (Renderer): Implementing EncounterXmlReportRenderer. This includes fetching the Form JSON Schema, parsing the hierarchy (Sections/Pages), and implementing the sorting logic to map Observations (Obs) to their correct visual position.	**4.5 MD**
Backend	Async Controller Layer: Implementing EncounterDataPdfExportController with a job management pattern (leveraging OpenMRS Reporting). Includes endpoints for Initiating Job, Checking Status (polling), and Retrieving File, plus timeout handling.	**2.0 MD**

Configuration	Universal XSLT: Developing the master XSLT template that dynamically handles nested sections, headers, and standard formatting rules (1.15 spacing, bolding logic) based on the XML stream.	**1.5 MD**
Frontend	Async UI & Selection: Implementing the "Print Context Modal" with state management for the asynchronous flow (Loading Spinner to Polling Logic to Download Button) and error handling (toasts for timeouts/failures).	**4.0 MD**
QA	Testing: Manual testing of complex forms (e.g., nested groups), unit testing of the JSON parser logic, and **performance testing** of the async flow, and regression testing of the existing module features.	**2.5 MD**

Management	Process: Documentation, Code Review, and Deployment coordination.	**1.0 MD**
Management	Additional effort needed for community alignment + missed scope/unforeseen activities	**3.0MD**

5.2 Total Estimated Effort

Total: ~18.5 Man-Days (MD)