Software Requirements Specification

For

Federation Architecture for Composed Infrastructure Services (FACIS)

Federation Architecture Pattern Decentralized Catalogue Management

FACIS.FAP_DCM

Conformance Language

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC 2119[1].

1. Executive Summary

The Federation Architecture Pattern Decentralized Catalogue Management (FAP DCM) enables trusted, cross-catalogue discovery and representation of digital service and resource descriptions within a federated environment. It is essential for providing unified search capabilities across decentralized service catalogues, ensuring **integration with Eclipse XFSC Orchestration Engine (ORCE) and Identity, Credential and Access Management (ICAM)**[2], and supporting operation in cross-domain pilots.
By introducing a multi-catalogue abstraction layer, it allows seamless cross-catalogue searches, service registration, and metadata harmonization while ensuring interoperability, transparency and supporting compliance requirements.
This FAP enhances service discoverability across multiple cloud offerings, data spaces, and domains and decouples catalogue management in multidomain eco systems. A key function is to keep trust through verifiable service and resource listings.
The individual effort to map different schemes into a local normalized representation according to the needs of the target group will be reduced by usage of AI supported mapping functions. A common meta data representation will be addressed for data products and services with a Data Catalog Vocabulary (DCAT)[3] and where applicable as well for cloud/edge services.

The purpose of this tender is to commission software development services for Lot “Federation Architecture Pattern Decentralized Catalogue Management”. The delivery of the software is provisionally by Q1 of 2026. The final clarification will be communicated after the award of the tender.

2. Background & Context

This FAP addresses one of the key challenges in federated ecosystems: fragmented and siloed service catalogues. Its purpose is to enable discovery, comparison, and integration of digital services like cloud and resources like product passport **across multiple catalogues** without centralizing them.
FACIS is part of the 8ra Initiative[4], also known as IPCEI-CIS (Important Project for Common European Interest – Cloud Infrastructure and Services). The 8ra Initiative is driving the development of a sovereign, interoperable, and secure Multi-Provider Cloud-Edge Continuum. With more than 120 partners from 12 EU Member States, the 8ra Initiative fosters open collaboration to create a resilient and scalable digital infrastructure tailored to Europe’s needs. By promoting open-source innovation, interoperability, and cross-border cooperation, the 8ra Initiative empowers industries, researchers, and policymakers to shape the next

[1] https://www.rfc-editor.org/rfc/rfc2119
[2] https://projects.eclipse.org/projects/technology.xfsc
[3] https://www.w3.org/TR/vocab-dcat-3/
[4] https://www.8ra.com/8ra-community/about-the-8ra-initiative/

generation of cloud and edge computing. Whether in AI, data spaces, telecommunications, or the industrial metaverse, the 8ra Initiative enables groundbreaking advancements free from vendor lock-in.
As a result, a huge mass of Services on Infrastructure, Platform, Software and assembled Business Layer will emerge and needs to be categorized and offered on a per domain or per federation filter. The FAP provides the core functionality to address these scenarios as a pre-defined blueprint with full functional implementation as Open Source for individual adoption and fine tuning.

3. Scope & Boundaries

3.1 In Scope

- Unified search across distributed service catalogues.
- Interoperable service listings and metadata exchange.
- Catalogue changes resulting from Asset Crawling + Mapping processes (e.g., creating a new asset version, creating a new asset, or deleting an asset due to a remote deletion)
- Integration with ORCE to support lifecycle management features like deployment hooks and updates. Further on ORCE includes key functionality like UI Builder, Workflow Capabilities and seamless integration of boundary systems via REST API. ORCE is based on Node-RED[5] and includes all baseline functions as well.
- Integration with ICAM to ensure secure access to services and resources. XFSC delivers a full stack for Authentication and Authorization (e.g., Keycloak), Credential Manager for Organisations and Individuals as well as Policy Engine for specific rules.[6]
- Support for cross-domain use cases, enabling service discovery across different administrative domains.

3.2 Out of Scope

- Direct edits or arbitrary updates to catalogue content outside the workflow of the Asset Crawler + Mapper.
- Publish/subscribe mechanisms beyond the XFSC Federated Catalogue's existing interface with the Gaia-X Credential Event Service (CES), i.e., no implementation of the original optional requirements G-FR-07 and G-FR-08 [FC.CCF.SRS].
- Ontology alignment (applied to schemas)
- Enforcing a global, unified schema across all catalogues.

[5] https://nodered.org/
[6] https://eclipse.dev/xfsc/traincd/traincd/

- Service instantiation or direct execution.
- Deep integration beyond metadata exchange and access control.

4. Conceptual Architecture

This system provides a unified architecture to search, aggregate, and present metadata from multiple distributed service catalogues. It supports interoperable service listings, metadata exchange, and integration with lifecycle and security management systems. The system aims to enable secure, cross-domain access to catalogue data while maintaining separation of concerns between data retrieval and data management.

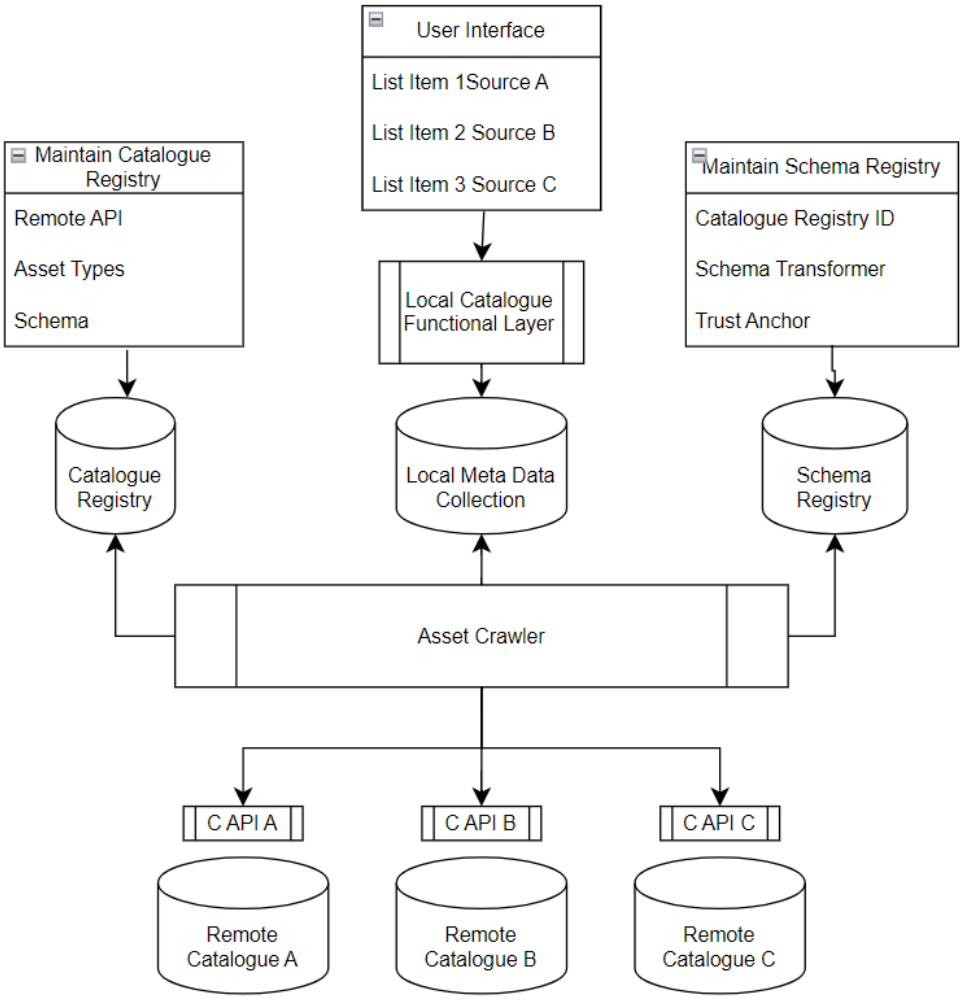

Figure 1 FAP DCM: Architecture Overview

FAP Components

- **Local Catalogue Functional Layer and User Interface (UI)**
    - Responsible for managing local metadata collection.
    - Facilitates user interactions including search and retrieval.
    - Presents unified catalogue views combining multiple sources.
- **Catalogue Registry and UI**
    - Maintains metadata for remote catalogue services.
    - Stores key attributes such as API endpoints, trust anchors, schema types, and supported query languages.
    - Provides UIs for managing catalogue entries.
- **Schema Registry and UI**
    - Maintains a model of the remote catalogues, from the perspective of the local catalogue: what we expect to find in remote catalogues, continuously updated according to what we found there.
    - Maintains mappings between local schema and remote schemas for each catalogue entity.
    - Stores trust anchors and schema transformers.
    - Provides administrative UI for managing schema relationships.
- **Asset Crawler**
    - On-demand crawler harvesting remote assets via catalogue APIs.
    - Transforms remote metadata into local schema representation using mappings.
    - Stores transformed metadata into the local meta data collection for efficient querying.
    - Provides administrative UI for initiating a harvest. This should enable the user to set the following parameters:
        - Remote Catalogue ID(s)
        - Query
        - Schema mapper

Key Components

- **XFSC Services**
    - **CAT (Catalogue)** – local data container for services and assets.
    - **ORCE** – orchestration hooks for resource lifecycle.
    - **AAS** – authentication and authorization service.
- **Data Flow**
    1. Users interact with the UI to perform unified searches or browse catalogue items.
    2. The Local Catalogue Functional Layer processes these requests, querying the Local Meta Data Collection.
    3. The Asset Crawler retrieves metadata from remote catalogues via respective APIs (C API A, B, C), transforming and updating the local metadata store.
    4. The crawler interacts with the Catalogue Registry and Schema Registry to ensure data consistency, proper schema transformation, and validation against trust anchors.
    5. Administrators manage catalogue and schema registries through their respective UIs.
- **Interfaces & Protocols**
    - **Remote APIs:** Communication with remote catalogues via defined APIs, typically RESTful or SOAP endpoints (e.g., API A, B, C).
    - **Internal APIs:** For interactions between the Asset Crawler, Local Catalogue Functional Layer, Catalogue Registry, and Schema Registry.
    - **User Interface API:** Supports querying and retrieval of metadata.
- **Schema and Data Models**
    - **Catalogue Entries:** Metadata structure representing service listings from remote catalogues.
    - **Local Meta Data Collection:** Stores transformed metadata in a unified local schema.
    - **Schema Mapping:** Defines relationships between local and remote schema attributes.
    - **Trust Anchors:** Security elements to verify claims and ensure data authenticity. In line with CAT-FR-CO-03 of the Federated Catalogue Enhancement

specification, the actual verification is performed by external services [FACIS.FCE.SRS].

- **Schema Transformers:** Logic for converting remote metadata formats into local schema formats.

- **Security Considerations**
  - **ICAM Integration:** Ensures secure authentication and authorization for accessing services and catalogues.
  - **Trust Anchors:** Used in Schema Registry for verifying service listing claims and metadata integrity.
  - **Secure API Access:** Remote API connections secured using tokens, certificates, or other authentication mechanisms.
- **Data Validation**

  Enforced at schema transformation stage to prevent invalid or malicious data.
- **Error Handling and Logging**
  - **Asset Crawler:** Implements retry logic and error logging for failures during remote crawling.
  - **Validation Errors:** Logged and reported for schema mismatches or security violations.
- **UI**

  Presents meaningful error messages for data retrieval failures.
- **Deployment and Maintenance**
  - **Catalogue Registry & Schema Registry:** Interfaces available for administrators to maintain remote catalogue definitions and schema mappings.
  - **Asset Crawler:** Runs on demand or on schedule to ensure up-to-date metadata.
  - **User Interface:** Updated regularly to support new catalogues or schema changes without disrupting user experience.

5. Functional Requirements

5.1 Asset Crawler + Mapper

FR-ACM-01 Initiating a Harvest

Priority: MUST HAVE
Description:
There MUST be a dedicated endpoint via which an authorized user can initiate the harvest of one or more remote catalogues with the following arguments:

- a reference to one, or more, or all remote catalogues, as registered in the Catalogue Registry according to FR-CR-01,
- a harvest scope, specified as a one-off argument or as a reference to a pre-configured harvest scope according to FR-ACM-02,
- whether schema mapping is to be performed.

Errors that prevent the harvest from being executed MUST be logged; these include

- the remote catalogue not being available, or
- the remote catalogue not being able to interpret the harvest scope, e.g., not being able to interpret the given query.

Each harvested asset MUST be imported into the local catalogue, jointly with the following provenance metadata:

- The unique identifier of the asset in the remote catalogue, if available
- A reference to a metadata record about the harvest, including
    - The unique identifier of the remote catalogue in the Catalogue Registry
    - The date and time of the harvest
    - The scope of the harvest according to FR-ACM-02.

If schema mapping is to be performed, the transformation configured in the Catalogue Registry for the respective remote catalogue MUST be applied to each harvested asset as specified in FR-SR-05 for the AI-driven transformation, in FR-SR-07 for the deterministic RDF mapping, or in FR-SR-08 for the hybrid strategy. The result of a successful mapping MUST be stored as an asset linked to the originally imported asset according to FR-ACM-04; errors encountered during the mapping MUST be logged. The audit trail of the transformation (cf. FR-SR-09) MUST be stored as additional provenance metadata.
Acceptance criteria:

1. Set up a Catalogue Registry with 2 remote catalogues C_1 and C_2.
2. Execute the harvesting of
    a. C_1,

   b. C_1 and C_2, and, after having updated assets in both C_1 and C_2,

   c. all of the stored remote catalogues (i.e., here, once more C_1 and C_2).

3. After each of (2a.), (2b.) and (2c.), query the local catalogue showing that the harvested assets are now stored in the local catalogue and that their metadata contain all the provenance information defined in this requirement.

Notes:

- The user MAY schedule recurring harvests by external mechanisms such as *cron*, which invoke the endpoint in defined intervals.
- After initiation, a harvest will run unattended, not expecting the user to intervene. Instead, the user MAY study the log and provenance metadata resulting from a requested harvest.
- It may be beyond the control of the Asset Crawler or Mapper whether, in the local catalogue, an imported asset may use the same identifier that it had in the remote catalogue. Especially for the case where this is not possible, we record that identifier separately.
- Schema mapping is not part of the acceptance criterion of this FR, as its correct implementation is validated by the acceptance criteria for the respective FRs for the Schema Registry.

FR-ACM-02 Harvest Scope

Priority: MUST HAVE
Description:
The user MUST be able to specify what should be harvested from a remote catalogue:

1. All assets of the given remote catalogue
2. All assets of a given type, as specified in FR-CR-02
3. As a generalization of (2.), if supported by the remote catalogue, only those assets that match a given query
4. All assets that have changed since a given date and time
5. All assets that have ever been imported from the given remote catalogue
6. All assets that were imported from the given remote catalogue in the last harvest (according to FR-ACM-01)
7. For (5.) and (6.), the variation MUST be supported to additionally include any new assets in the remote catalogue, as in (4.)

8. All assets in the scope of the last harvest (including, e.g., beyond (6.), new assets of the type specified according to (2.)).

Acceptance criteria:

1. Set up one remote catalogue, which is capable of being harvested in all ways defined in this requirement, and which contains two assets A_1 and A_2 with last change dates 2025-02-15 and 2025-05-28.
2. Harvest the remote catalogue once in ways (1.), (2.), (3.), and (4.) defined above, specifying the date 2025-03-29.
3. In the remote catalogue, update asset A_2 and create a third asset A_3, having the same type as A_1.
4. Harvest the remote catalogue once more in ways (5.), (6.), and (7.) defined above.
5. After each harvest specified in acceptance criteria steps (2.) and (4.), show that the assets stored in the local catalogue match the given harvesting parameters, then revert the catalogue to the state before that harvest.

FR-ACM-03 Lifecycle of Imported Assets

Priority: MUST HAVE
Description:
Upon initiating a harvest, the user MUST be able to define how the lifecycle of remote assets should be reflected locally, i.e.:

- Whether a new version of a remote asset should be realized as a new version of the corresponding local asset, if it exists, or whether a separate local asset should be created,
- Whether the deletion of a remote asset should result in the deletion of the local asset, or whether the local copy should be retained.

Acceptance criteria:

1. Set up a remote catalogue with one asset. Harvest that asset.
2. Update the asset in the remote catalogue. Then, execute two harvesting processes with different settings of the remote asset lifecycle parameter:
    a. For the first execution, a separate asset should be created in the local catalogue.
    b. For the second execution, the existing asset in the local catalogue should be updated with a new version.
3. Delete the asset in the remote catalogue, and once more execute two harvesting processes with different settings of the remote asset lifecycle parameter.

   a. For the first execution, the local catalogue should remain unchanged
   b. For the second execution, the corresponding asset in the local catalogue should be removed.
4. After each step, run a query to demonstrate that the local catalogue has the intended state.

Note:

As it is NOT REQUIRED to implement a publish/subscribe mechanism, by which the local catalogue might be notified of the deletion of an asset in a remote catalogue, the only way to find out that an asset has been deleted is that it does not occur in subsequent harvests anymore.

FR-ACM-04 Linked Assets

Priority: MUST HAVE

Description:

To ensure that, for any asset imported from a remote catalogue, both its original form and the transformed form resulting from schema mapping remain accessible, both forms MUST be stored in the local catalogue in a way that does not cause conflicts, i.e.:

- As the result of a regular query, the local catalogue MUST only return the transformed asset, unless explicitly asked for the original asset before transformation.
- Throughout the lifecycle of the imported asset (cf. FR-ACM-03), both the original form and the transformed form must be handled together.

Depending on the local catalogue's implementation, this functionality MAY be realized by keeping the original form of an imported asset in a separate namespace of the asset store, or by only keeping the transformed form in the asset store and encoding the original form into special metadata.

Acceptance criteria:

1. Run a harvest with successful schema mapping according to FR-ACM-01, which results in at least one asset imported.
2. Demonstrate queries that handle both the original form and the transformed form of the imported asset correctly, i.e.:
   a. Run a regular query that demonstrates that only information about the transformed form of the asset is considered.
   b. Run a query that explicitly asks for the original asset.
3. Demonstrate that the lifecycle of both the original form and the transformed form is handled correctly, i.e.:
   a. Update the asset in the remote catalogue, re-run the harvest of step (1.), ensuring that the updated asset is imported into the local catalogue.

Demonstrate that there are new versions of both the original form and the transformed form of the asset.

b. Delete the asset from the local catalogue. Demonstrate that both the original form and the transformed form have been deleted.

FR-ACM-05 Crawling References

Priority: MUST HAVE

Description:

For the case that assets of one remote catalogue C_1 reference assets stored in another catalogue C_2 (e.g., hierarchical service dependencies), the Asset Crawler MUST be configurable to automatically resolve such links during a harvest, or not. The following reference types MUST be supported:

- URLs (Linked Data style)
- DIDs [DID]

Acceptance criteria:

1. Set up two Catalogues C_1 and C_2 with one asset each, A_1 and A_2, both of type *dcat:Resource*. A_1 references, via *dcat:qualifiedRelation* / *dcat:Relationship*, to A_2.
2. Demonstrate two harvests from C_1:
    a. One in which only A_1 is retrieved
    b. One in which A_2 is retrieved as well, because the link to C_2/A_2 is crawled.

5.2 Schema Registry

FR-SR-01 Storing Remote Schemas

Priority: MUST HAVE

Description:

The Schema Registry MUST provide the possibility to store, for reference, any remote schemas that the local catalogue's administrator has become aware of. These schemas MUST be kept separately from the schemas used by the local catalogue for validating local assets. For each schema, the following metadata SHALL be maintained: the unique ID(s) of those remote catalogues in the Catalogue Registry (cf. FR-CR-01), in which this schema is used

Acceptance criteria:

1. Set up a schema-aware local catalogue with one schema.
2. Upload one "remote schema" S to the Schema Registry. It is NOT REQUIRED to actually run a remote catalogue for this demonstration.

3. Demonstrate that, from the local catalogue, schema S is not accessible for purposes that schemas are typically used for, e.g., validating assets.

Notes:

- The rationale for keeping the remote schemas separately from the local schemas is that local assets are not necessarily valid against the remote schemas and vice versa.
- The rationale for emphasizing the separation of local and remote schemas is that the Schema Registry MAY be realized by storing schemas in a schema-aware local catalogue, but they MUST NOT interfere with the schemas regularly stored there.
- The Asset Crawler + Mapper is NOT REQUIRED to harvest schemas from a remote catalogue, as state-of-the-art catalogues do not typically expose their schemas. Nevertheless, there are exceptions, such as the XFSC Federated Catalogue [FC.CCF.SRS].

FR-SR-02 Transformation Strategy Selection

Priority: MUST HAVE
Description:
For each remote catalogue, one of the following transformation strategies MUST be configurable:

- AI-Driven: LLM transforms asset using prompts
- Deterministic RDF: schema-selective RDF Mapping
- Hybrid: first try AI, but fallback to RDF on failure

The selected strategy MUST be configured in the Catalogue Registry (cf. FR-CR-01). Changing the strategy for one remote catalogue MUST affect subsequent harvests from that remote catalogue but MUST NOT affect previously imported versions of assets from that remote catalogue.

Acceptance criteria:

1. Configure two remote catalogues, each with a different transformation strategy.
2. For one remote catalogue C_1, change the strategy.
3. Harvest assets from C_1; demonstrate that the new strategy is being applied according to the acceptance criteria for the respective FR(s).

FR-SR-03 Prompt Storage as Versioned Assets

Priority: MUST HAVE

Description:

Prompts to guide the AI-driven transformations of catalogue assets MUST be stored in Schema Registry as versioned assets with the following information:

- *ID* (UUID)
- *version* (major.minor.patch)
- *status* (draft/active/deprecated/archived)
- s*ource schema ID* (foreign key to remote schema, cf. FR-SR-01) and *target schema ID* (foreign key to local schema).
- *prompt template ID* (foreign key to prompt text with variable placeholders, cf. FR-SR-04)
- *examples* (substituting the *{EXAMPLES}* prompt template variable with a concrete example for few-shot learning, e.g., showing how to transform complex data structures)
- *constraints* (substituting the *{CONSTRAINTS}* prompt template variable with a concrete textual specification of transformation rules, e.g., indicating that some metadata terms should, or should not, be mapped to their closest DCAT matches)
- *created at*, *modified at* (timestamps)
- *author* (string)

The constraint MUST be enforced that there can only be one active prompt per source-target schema pair.

Acceptance criteria:

1. In the Schema Registry, create a prompt P_1 for mapping between a source schema S_s and a target schema S_t.
2. Create a new version of this prompt, making it active, and deprecating the initial version.
3. Demonstrate that both the new version and the initial version can be retrieved.
4. Create a new prompt P_2 for mapping between S_s and S_t. Try to make it active. Expect the operation to fail.
5. Delete the prompt.

FR-SR-04 Prompt Templates and Variables

Priority: MUST HAVE
Description:
Templates for prompts (cf. FR-SR-03) MUST be stored as versioned assets with the following information:

- *ID* (UUID)
- *version* (major.minor.patch)
- *text*

Prompt templates MUST support the following variable placeholders:

- *{SOURCE_ASSET}* – Complete remote asset (harvested from the remote catalogue)
- *{SOURCE_SCHEMA}* – Remote schema definition (retrieved from the Schema Registry)
- *{TARGET_SCHEMA}* – Local schema definition (retrieved from the local catalogue)

In addition, prompt templates MUST support the following optional variable placeholders:

- *{EXAMPLES}* – Optional example transformations (specified in the prompt that instantiates this template according to FR-SR-03)
- *{CONSTRAINTS}* – Optional transformation rules (specified in the prompt that instantiates this template according to FR-SR-03)

These variables are intended to be substituted at runtime before sending the prompt to the LLM (cf. FR-SR-05).

Acceptance criteria:
None for this FR on its own, but the correct implementation of this FR will be validated via FR-SR-05.

FR-SR-05 AI-Driven Transformation Execution

Priority: MUST HAVE
Description:
The AI Driven schema mapping, which is triggered during a harvest when referenced by the respective remote catalogue configuration (cf. FR-CR-01) MUST follow this execution pipeline:

1. Retrieve active prompt from Schema Registry (cf. FR-SR-03)
2. Substitute template variables (cf. FR-SR-04)
3. Send to the complete prompt to the LLM Parse the LLM's response.
4. Validate the output against the local schema.

5. Store the result including, in case of successful validation, the transformed asset, and the following metadata:
    - prompt version,
    - model used,
    - timestamp,
    - duration,
    - status,
    - errors.

Acceptance criteria:

1. Configure a remote catalogue with AI-driven transformation.
2. Perform a harvesting operation.
3. Show that the result conforms to the local schema.
4. Query the metadata to show that the selected LLM parameters were correctly set.
5. Repeat steps (2.) to (4.) with varying LLM parameters and show that they impact the result and show that prompts with missing values for the *{EXAMPLES}* and *{CONSTRAINTS}* variables in prompt templates are handled without errors.
6. Perform a harvesting operation that enforces a timeout.
7. Show that the failures are correctly logged.

Note:
This requirement MAY be realized using a framework such as LangChain[7], which provides prompt template management, variable substitution, and multi-provider LLM integration.

FR-SR-06 Deterministic Schema-selective RDF Mapping

Priority: MUST HAVE
Description:

- The deterministic mapping of RDF assets from one schema to another schema assumes that a subset of the RDF triples in an asset imported from a remote catalogue constitute a valid asset in the local catalogue because both catalogues have a common understanding of a standard vocabulary, e.g., DCAT. Thus, it MUST be possible to configure the following:http://www.w3.org/ns/dcatNamespace(s) of metadata vocabularies to preserve (e.g., *http://www.w3.org/ns/dcat#*)
- Optional reference to a SHACL shape (stored as a schema in the local catalogue) to validate the retained triples.

Mark as incomplete if validation fails.

[7] https://www.langchain.com/

Acceptance criteria:

None for this FR on its own, but the correct implementation of this FR will be validated via FR-SR-07.

FR-SR-07 Deterministic RDF Mapping Execution

Priority: MUST HAVE
Description:
The deterministic RDF schema mapping MUST follow this execution pipeline:

1. Parse RDF asset to triples
2. Retain all triples whose predicate is in one of the namespaces to be preserved; discard any other triples.
3. If a SHACL shape is configured, take the preserved triples as a data graph and validate it against the SHACL shape. In case of successful validation, or in case no SHACL shape was configured, keep the preserved triples as the transformed asset; otherwise raise an error.

Acceptance criteria:

1. Configure a remote catalogue to use deterministic RDF mapping:
    a. with SHACL validation, and
    b. without SHACL validation
2. In the remote catalogue, create one RDF asset A_1 that is valid against the given SHACL shape (after only retaining triples with predicates from the namespaces to be preserved), and another one A_2 that is not.
3. Perform a harvest from that catalogue.
4. Run a query against the result:
    a. If SHACL validation is turned on, show that
        - A_1 was mapped by discarding triples from unknown namespaces, and
        - A_2 was not mapped.
    b. If SHACL validation is turned off, show that both A_1 and A_2 were mapped by discarding triples from unknown namespaces.

Note:
Triples discarded during this transformation will not be part of the transformed asset and thus not queryable. However, they will remain accessible in the original form of the remote asset, which is also stored in the local catalogue according to FR-ACM-04.

FR-SR-08 Hybrid Mapping Strategy with Automatic Fallback

Priority: MAY HAVE
Description:
There MAY be a hybrid mapping strategy, which falls back to the Deterministic RDF mapping (FR-SR-06, -07) in case the AI-Driven mapping is not successful for at least one of the following reasons:

- LLM timeout/network error,
- unparseable output,
- validation failure,
- rate limit.

1. The execution sequence is as follows: Attempt AI-Driven transformation
2. On failure, fall back to Deterministic RDF mapping.

Acceptance criteria:

1. Configure a remote catalogue.
2. Perform a harvesting operation using the AI-driven mode.
3. Set it up in a way that is not successful.
4. Verify from the logs that it has been attempted.
5. Demonstrate that the unsuccessful AI-driven mapping triggered the fallback, resulting in a deterministic RDF mapping.

FR-SR-09 Transformation Audit Trail

Priority: MUST HAVE
Description:
The Schema Registry MUST store a tamper-proof record per mapping, which is exportable as CSV and JSON. It MUST contain:

- transformation ID
- asset IDs (remote/local)
- catalogue ID
- timestamps
- strategy
    - For AI: prompt ID and version
    - For deterministic RDF: preserved namespaces
- results of schema validation
- status
- errors
- duration

- metadata about the validation as specified in the XFSC Federated Catalogue requirements CAT-FR-CO-02 and CAT-FR-CO-05 [FACIS.FCE.SRS].

Records SHOULD be queryable by:

- asset ID
- catalogue ID
- prompt version
- status
- date range

Acceptance criteria:

1. Set up a remote catalogue.
2. Perform a harvesting operation.
3. Show that a record of the mapping is stored in the Schema Registry.
4. Export the record once as CSV and once as JSON.
5. Show that it is not possible to store a modified version of the same record in the Schema Registry.
6. Show that it is possible to query the records by the parameters defined in this requirement.

FR-SR-10 Prompt Testing Interface

Priority: MUST HAVE
Description:
The Schema Registry MUST have a testing UI, which provides controls for:

- Prompt version selection
- Sample asset input
- Dry-run execution (no persistence)
- Output view and comparison

For test cases, the following data SHOULD be stored with each prompt:

- sample input
- expected output
- description

Acceptance criteria:

In the UI, demonstrate the following functionality:

- It should be accessible from prompt management UI.

- It has a side-by-side source/result display.
- The run test cases can be saved and reused.
- It offers a setting to support batch testing.

FR-SR-11 LLM Configuration

Priority: MUST HAVE
Description:
The Schema Registry MUST support storing configurations of LLMs with the following parameters:

- ID (UUID)
- LLM provider (e.g., OpenAI, Anthropic, Ollama, custom)
- Model identifier
- Temperature
- Token limit
- Timeout

Acceptance Criteria:

1. Create two different LLM configurations L_1 and L_2.
2. Set up two remote catalogues C_1 and C_2, each of which uses AI-driven transformation using L_1 and L_2, respectively.
3. Demonstrate the results of a harvest and the subsequent mapping of harvested assets. You MAY use different prompts that highlight the differences of the two LLMs.

FR-SR-12 Multi-Model Provider Support

Priority: MAY HAVE
Description:
The providers OpenAI, Anthropic, and Ollama SHOULD be supported, as well as custom endpoints.
Per provider, it SHOULD be possible to configure the following options:

- endpoint URL
- authentication
- credentials (encrypted)
- model ID

- rate limits
- timeouts

It SHOULD be possible to reference a provider on each of the following levels:

- per prompt (by extending FR-SR-03)
- per catalogue (by extending FR-CR-01)
- as a global system default.

The model to be applied SHALL be selected according to the following precedence rule: prompt level > catalogue level > system level.

Acceptance criteria:

- Demonstrate a setup, in which multiple providers are configured simultaneously.
- Demonstrate the selection of a provider selection per prompt, per catalogue, and as a global system default.
- No code changes for provider selection.
- Demonstrate that credentials are stored encrypted.
- Demonstrate that rate limits are enforced.
- Demonstrate how usage is tracked per provider and per model.

FR-SR-13 Batch Re-transformation

Priority: MAY HAVE
Description:
It SHOULD be possible to have a re-transformation of previously imported assets triggered by any of the following events:

- changing the transformation strategy in a Catalogue Registry entry (cf. FR-CR-01),
- For AI-driven transformation: a new version of a prompt or of an LLM
- For deterministic RDF mapping: changes to the namespaces and target schema.

The scope of assets, to which this re-transformation is applied, SHOULD be configurable:

- All assets
- All assets of a given type
- All assets harvested from (a) given remote catalogue(s)

The lifecycle of assets according to FR-ACM-03 MUST be respected.

There SHOULD be a dry-run mode, and an interactive mode with progress tracking and the possibility to cancel.

Acceptance criteria:

- Demonstrate the initiation of the interactive mode via a UI or an API call.
    - Show that the progress is visible during execution.
    - Cancel the execution.
- Original assets preserved (versioning).
- Asset lifecycle respected: show that their previous versions are still accessible.
- Show that a summary report has been generated.
- Demonstrate a failed re-transformation and how it is successfully re-tried after fixing the transformation configuration that caused the problem.
- Individual retry supported.

5.3 Catalogue Registry

FR-CR-01 Configure Remote Catalogues

Priority: MUST HAVE
Description:
It MUST be possible to configure remote catalogues, from which assets can be crawled, according to the following communication protocols / APIs:

- OAI-PMH
- DCAT as Linked Data: a URL that resolves to a *dcat:Catalog*
- Query interface: an endpoint to which a query is sent

For each such remote catalogue the configuration MUST include the following information:

- a unique identifier
- the identifier of communication protocol, as listed above
- in the case of communication with a query interface:
    - the query endpoint(s)
    - for each query endpoint, the supported query language(s)
- a MIME type identifying the expected result, e.g., *application/sparql-results+json* for a SPARQL result set in JSON format [SPARQL].

- the desired transformation strategy for schema mapping according to FR-SR-02, if any, and:
    - If the AI-driven or hybrid strategy is selected, the ID of an active stored prompt to guide the AI-driven transformation (cf. FR-SR-03) and of an LLM to execute the prompt (cf. FR-SR-11).
    - If the deterministic RDF mapping is selected, the namespace(s) of URIs to preserve and, optionally, the identifier of a SHACL shape in the local catalogue to validate the mapping result (cf. FR-SR-06).

Acceptance criteria:
For each of the three required communication protocols, demonstrate the harvest from a catalogue containing at least two assets according to FR-ACM-01. After each harvest, demonstrate its success by running an appropriate query to retrieve information from the assets harvested, then revert the asset store to the state before the harvest. This MAY be realized as three harvests from less than three catalogues, if one catalogue speaks more than one communication protocol.

FR-CR-02 Configure Asset Types

Priority: MUST HAVE
Description:
There MUST be a registry of asset types that we would like to import to the local catalogue and and a mapping of asset types expected to be found in the remote catalogues onto these. The syntax of asset type identifiers (e.g., plain words or URIs of classes defined in an ontology) MAY vary based on the specification of the remote catalogue.

- In case the identifier of an asset differs between local and remote catalogue(s), the asset type registry MUST contain a mapping of each remote asset type identifier to the local type,
- In the case that an asset type is identified in the same way in the local catalogue and in all remote catalogues configured in the Catalogue Registry, it MUST be possible simply add the plain asset type identifier as an entry without an explicit mapping.

Acceptance criteria:
Show the registry of asset types, containing both mappings and plain entries.

Note:
There is no standard way of obtaining the types of assets of remote catalogues. The local administrator MAY extract them from a harvest according to FR-ACM-01. Depending on the communication protocol spoken by the remote catalogue (cf. FR-CR-01), a harvest MAY include explicit type information; this is the case, e.g., with DCAT and SPARQL query results. For a remote catalogue with a query interface, it MAY alternatively be possible to query for all asset types. Another option is to consult the remote catalogue's documentation.

FR-CR-03 Catalogue API Mapping

Priority: MAY HAVE
Description:
To enable the harvesting of assets from remote catalogues by type (case (2.) of FR-ACM-02) in a broader range of scenarios, mappings from asset types (defined according to FR-CR-02) to remote catalogue API requests MAY be pre-configured, e.g., mapping the asset type *https://w3id.org/gaia-x/development#ServiceOffering* to an API request like *GET https://remote.catalogue/assets?type=ServiceDescription*. Such API mappings SHOULD be AI-driven, as long as there is no deterministic implementation. The prompts for guiding API mappings SHOULD be managed like the prompts for schema mapping (cf. FR-SR-03), referencing, instead of schemas, the target remote catalogue according to FR-CR-01.

Acceptance criteria:

1. Set up a remote catalogue that has no query API.
2. In the local catalogue, define an asset type *T* and, if required for interoperability with the remote catalogue, a mapping to the remote asset type *m(T)*.
3. In the remote catalogue, create an asset of type *m(T).*
4. Harvest the asset.
5. Query the local catalogue to demonstrate that the harvested asset has been imported.

5.4 Access Control

FR-AC-01 User Roles

Priority: MUST HAVE
Description:
It MUST be possible to define different User Roles, which allow or deny accessing the functionalities of one / some / all of the following:

- the Asset Crawler: initiation of a harvest (FR-ACM-01), Harvest Scope CRUD operations (FR-ACM-02), configuring lifecycle (FR-ACM-03) and crawling (FR-ACM-05) options
- CRUD operations in the Schema Registry (FR-SR-01, -03)
- CRUD operations in the Catalogue Registry (FR-CR-01, -02, and optionally -03).

Acceptance criteria:

1. For each of the permissions listed above, assign to a user a role that has, or does not have, the respective permission.
2. Demonstrate the user's attempt of executing the respective action. Expect it to be denied when the user's role has not been granted the necessary permission.

5.5 General Interoperability

FR-GI-01 Discoverable Local Catalogue API

Priority: MUST HAVE

Description:

The local catalogue MUST expose its machine-readable API documentation in a way that enables it to be discovered,according to [RFC.9279].

Acceptance criteria:

Implement RFC 9279 in the XFSC Catalogue and demonstrate a request to */.well-known/api-catalog*.

6. Technical Architecture

6.1 Flows of Interaction with the Components

The following UML sequence diagrams show two typical flows of interaction with the components of the DCM FAP:

- Figure 2 shows the harvest of a remote catalogue without mapping. After the update of an asset in the remote catalogue, it is harvested once more.
- Figure 3 shows the harvest of a remote catalogue with AI-driven mapping of the harvested assets into the target schema of the local catalogue.

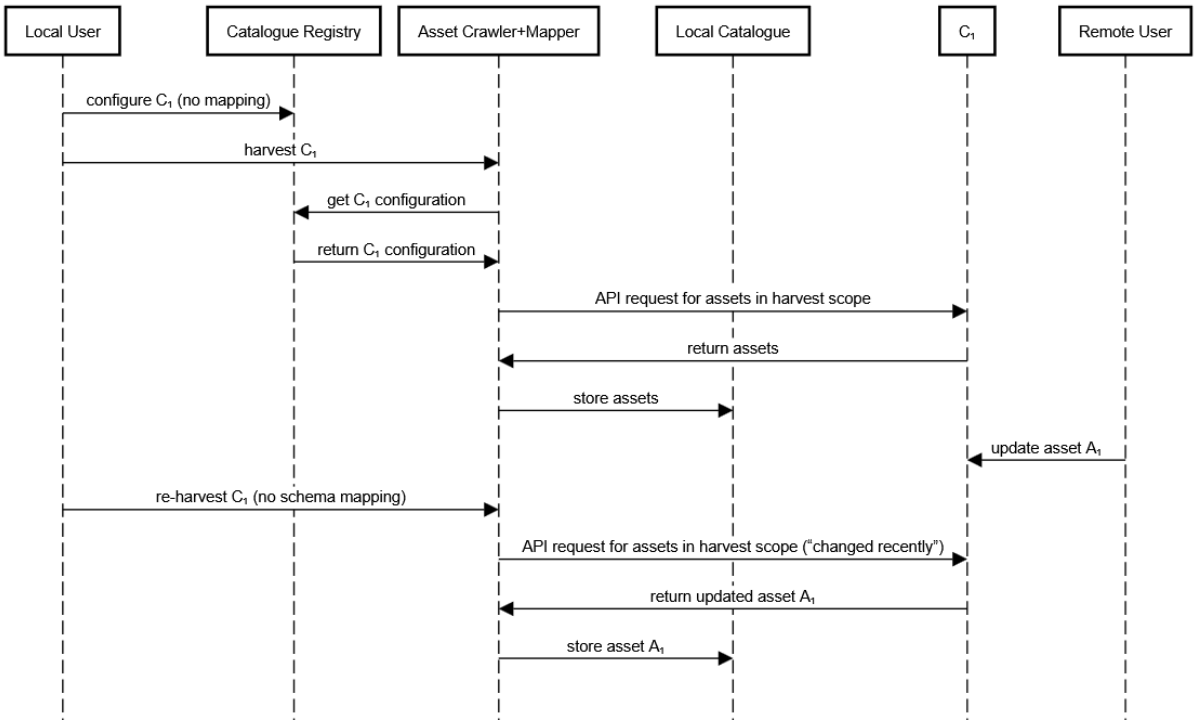

Figure 2 Harvest and Re-harvest without Mapping

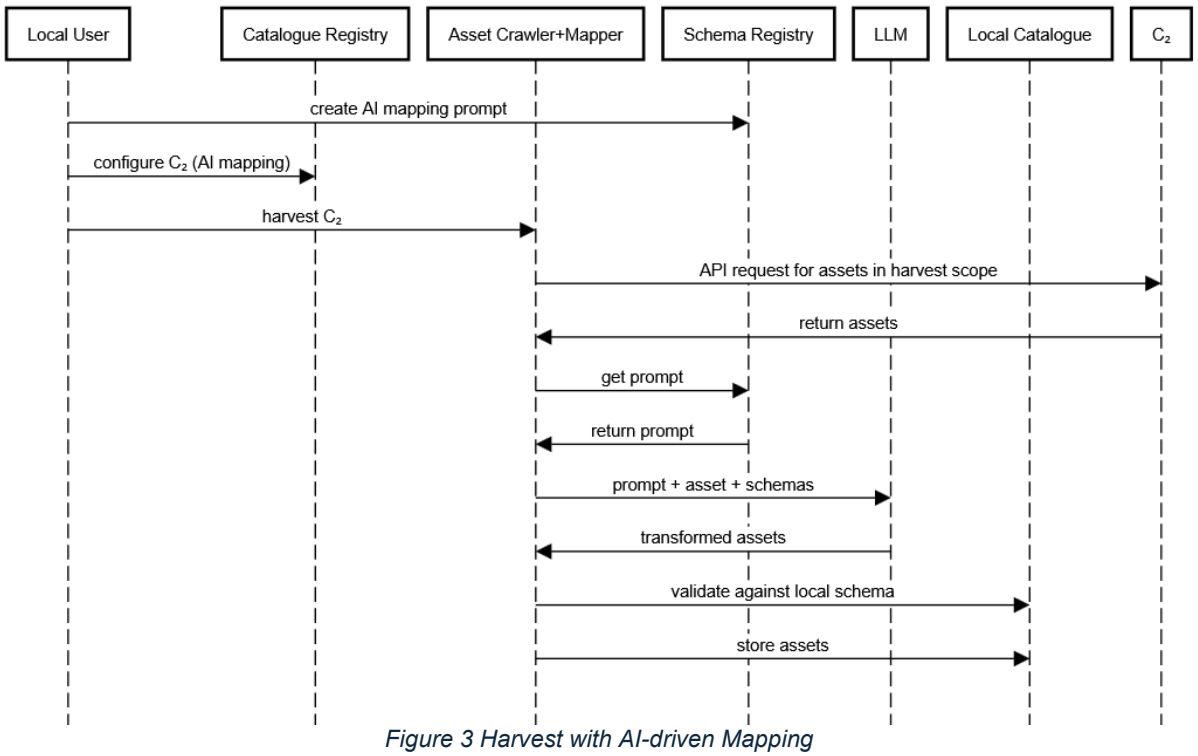

Figure 3 Harvest with AI-driven Mapping

6.2 ORCE Integration Requirements

This FAP MUST be implemented on top of the XFSC Orchestration Engine. ORCE provides the workflow runtime, UI generation layer, and integration capabilities required for configuring catalogue sources, executing harvest operations, managing schema transformations, and supporting administrative controls.

ORCE serves as the execution environment for all user facing and automated logic of the FAP DCM. The implementation MUST follow the principles described below.

6.2.1 UI Generation with ORCE UI Builder

All user interfaces of the FAP DCM MUST be created using the ORCE UI Builder technology. The UI Builder generates dynamic UIs based on ORCE flows and supports the use of modern frontend frameworks such as Vue.js and compatible UI component libraries.

The following UIs MUST be implemented using ORCE UI Builder:

- Schema Registry UI
- Catalogue Registry UI
- Asset Crawler + Mapper UI
- Prompt Management UI
- Prompt Testing Interface

- Administrative UI, including user and role management
- Monitoring pages

Using the UI Builder ensures consistency with other FACIS components and avoids manual HTML based UI creation where ORCE already provides structured rendering capabilities.

6.2.2 Use of ORCE Standard Components

Where ORCE provides standard nodes or UI components for a specific interaction pattern, these MUST be used. Examples include:

- Multi step wizards
- Standard form layouts
- Record tables with pagination
- Editor panels
- Action buttons connected to flows
- Validation and error elements

Whenever a UI matches an existing component type supported by ORCE, the implementation MUST rely on those built in components for consistency and maintainability.

6.2.3 UI Structures Without Available ORCE Components

If a required interface does not have a corresponding ORCE component, the implementation MUST follow the structural and interaction guidelines defined in the Reference FAP.

This includes:

- Layout structure
- Header and metadata placement
- Form grouping conventions
- Standard validation patterns
- Interaction flow patterns

6.2.4 Workflow Orchestration Using ORCE

All operational processes of FAP DCM MUST be implemented as ORCE flows. This includes:

- Harvest initiation
- Execution of schema transformations, including the recording of transformation audit trails

- Lifecycle processing for imported assets
- Schema validation and prompt testing
- Re-transformation workflows
- Configuration CRUD operations
- Access control operations defined in FR-AC-01

6.2.5 Integration Requirements

The FAP MUST integrate with ORCE as follows:

- ORCE flows invoking REST endpoints exposed by FAP DCM components
- User interfaces triggering ORCE actions through UI Builder
- Background processes executed through ORCE flows
- ORCE based access control enforcing roles defined in FR-AC-01
- Workflow status and logs managed through ORCE runtime features

7. Security & Trust

Transport Security

- All endpoints MUST use TLS 1.2 or higher
- Certificates managed via cert-manager in Kubernetes deployments
- API calls to LLM service MUST use TLS 1.3.

Identity & Authentication

- Organizations identified by W3C DIDs (did:web or did:key methods)
- Single Sign On to access Graphical User Interfaces using OAuth/OIDC using did:web identities.

Authorization & Policy Enforcement

User roles enforced (cf. FR-AC-01).

Data Protection

- Data at rest MAY be encrypted
- Data in transit encrypted via TLS/SASL
- Secrets stored in external KMS (HashiCorp Vault or AWS Secrets Manager)
- PII handling per GDPR data minimization principles.

Audit & Observability

- Audit trail includes LLM prompts/responses and schema validation results for traceability
- Logs aggregated in ELK stack or Loki for querying
- Metrics exported to Prometheus (request rates, latencies, error rates).

8. Deployment & Operations

8.1 Deployment Profiles (Exemplary)

Generally, it SHOULD be possible to define a set of default catalogues, which are pre-loaded into the Catalogue Registry of a new deployment of this FAP.

8.1.1 Docker Compose (Development)

Single-host deployment for local development and testing:

- Backend (FastAPI connector)
- Frontend (React UI)
- A graph database, e.g., Neo4j or an RDF triple store

8.1.2 Kubernetes (Production)

Multi-zone, highly available deployment:

- Helm charts for all services
- Separate namespaces per trust zone
- Horizontal Pod Autoscaling for connector and ingest services
- Ingress with TLS termination
- External Secrets Operator for KMS integration

8.2 Operational Requirements

8.2.1 Availability

- Control plane: 99.9% uptime SLA
- Data plane: 99.95% uptime SLA
- Health checks at /health and /ready endpoints
- Circuit breakers for external dependencies

8.2.2 Scalability

- Connector scales horizontally (stateless)
- 10 remote catalogues with 1,000 assets each can be harvested in parallel, including AI-driven schema mapping.

8.2.3 Monitoring & Alerting

- Prometheus metrics for all services
- Grafana dashboards for operational visibility
- Alertmanager for SLA breach notifications
- Distributed tracing with Jaeger.

9. Standards & Protocols

FAP DCM adheres to

- **W3C DID/VC:** Decentralized identifiers and verifiable credentials for asset provenance.
- **OAI-PMH**: Open Archives Initiative Protocol for Metadata Harvesting
- **DCAT v3**: Data Catalogue Vocabulary
- **SHACL**: RDF graph validation language
- **JSON-LD:** Linked data for standardized service metadata.
- **OpenAPI**: Service discovery and integration API.

10. Validation & Acceptance Criteria

10.1 Protocol Conformance

1. All endpoint descriptions validate against the OpenAPI specification.
2. Other than that, protocols are, where applicable, referenced by Functional Requirements, e.g., OAI-PMH by FR-CR-01, and thus to be validated and accepted as specified below for Functional Tests.

10.2 Functional Tests

For each Functional Requirement, acceptance criteria have been defined. The Requirements Traceability Matrix in Section 11 below summarizes these.

10.3 Security Tests

1. TLS certificate validation passes for all endpoints
2. Invalid VC presentation rejected during remote connect
3. Tampered HMAC signature rejected on HTTP data access if applicable
4. Data access: Authorization rules are followed during data retrieval via REST interface

10.4 Performance Tests

5. Local Catalog request responds < 500ms
6. HTTP data fetch completes < 2s for 100 local asset readings
7. Kafka throughput sustains 10,000 msg/sec per topic
8. UI access: < 1 sec

10.5 Operational Tests

9. **Mandatory use of XFSC Orchestration Engine for all operational workflows**
   All deployment, configuration, and runtime processes of the FAP DCM MUST be executed using ORCE flows. This includes UI actions, background processing, lifecycle operations, and integration logic.
10. **Deployment starts successfully**
    The development deployment MUST start without errors and all services MUST reach a healthy state.
11. **Kubernetes Helm chart deploys without errors**
    The production setup MUST deploy successfully using the provided Helm charts and all pods MUST become ready.

11. Requirements Traceability Matrix

Req ID	Component	Interface/API	Test Case
FR-ACM-01	Asset Crawler + Mapper: Harvest Initiation	OAI-PMH, DCAT / Linked Data, arbitrary Query languages as supported by remote catalogues via HTTP endpoints	Initiate simple harvesting operation of remote catalogue(s).
FR-ACM-02	Asset Crawler: Harvest Scope	arbitrary Query languages as supported by remote catalogues via HTTP endpoints	Conduct several harvesting operations with varying definitions.
FR-ACM-03	Asset Crawler: Asset Lifecycle	–	Harvest remote catalogue and validate different lifecycle settings, namely (1) versioning and (2) deletion.
FR-ACM-04	Asset Crawler + Mapper: Asset Linking	–	Harvest remote catalogue and show the linking and lifecycle handling of original and transformed asset.
FR-ACM-05	Asset Crawler: Referenced Assets	URLs, DIDs	Retrieve a remote asset from a catalogue C_1 that contains links to another catalogue C_2.
FR-SR-01	Schema Registry: Schema Storing	–	Upload a remote schema to the Schema Registry and validate it is not accessible from local catalogue for internal purposes, such as validating assets.
FR-SR-02	Schema Registry: Mapping Strategy	–	Harvest a remote catalogue using different transformation strategies; details to be validated per strategy.

FR-SR-03	Schema Registry: Prompt Storage	–	Create a new AI prompt for a mapping between two schemas; basic lifecycle.
FR-SR-04	Schema Registry: Prompt Templates	–	See FR-SR-05
FR-SR-05	Schema Registry: AI-Driven Transformation	–	Perform a harvesting operation on a remote catalogue using AI-driven transformation. Do this with varying LLM parameters.
FR-SR-06	Schema Registry: RDF Transformation Definition	–	See FR-SR-07
FR-SR-08	Schema Registry: Transformation Fallback	–	Showcase automatic fallback on deterministic RDF mapping, when AI-driven transformation is unsuccessful.
FR-SR-09	Schema Registry: Transformation Audit	CSV, JSON	Store the record of a mapping in an immutable way, then query and export it.
FR-SR-10	Schema Registry: Prompt Testing	–	Showcase UI that enables the testing of prompts.
FR-SR-11	Schema Registry: LLM Configuration	–	Create and store a LLM configuration in the Schema Registry.
FR-SR-12	Schema Registry: Multi Model Support	–	Perform a harvesting operation on a remote catalogue with multi-model provider support enabled.

FR-SR-13	Schema Registry: Batch Transformation	–	Change the prompt, transformation strategy or error correction and show that the already existing assets are transformed accordingly.
FR-CR-01	Catalogue Registry: Configure Catalogue	–	Perform a harvesting operation for each of the required communication protocols.
FR-CR-02	Catalogue Registry: Configure Types	URIs	Create and store a mapping of asset types from a remote to a local schema. Then perform a harvesting operations showcasing the correct mapping from remote to local assets.
FR-CR-03	Catalogue Registry: API Mapping	OpenAPI	Configure a mapping from an asset type to a API request and perform a harvesting operation utilizing this mapping.
FR-AC-01	Access Control: User Roles	–	Showcase the different User Roles by performing CRUD operations for the Asset Crawler, Schema Registry and Catalogue Registry. These should fail, if the user does not have the correct role.
FR-GI-01	General: Discoverable Catalogue	RFC 9279	Show that the local catalogue exposes its machine-readable API documentation in a discoverable way according to [RFC.9279].

12. Appendices

Appendix A: Glossary

Term	Full Name	Definition
CES	Gaia-X Credential Event Service	A common publication/subscription service that allows the Gaia-X catalogues of the GXDCH to be notified about new, updated, and revoked credentials [CES]
CRUD	Create, Read, Update, Delete	Four basic operations (actions) of persistent storage.
DID	Decentralized Identifier	W3C standard for decentralized digital identities [DID]
OAI-PMH	The Open Archives Initiative Protocol for Metadata Harvesting	A low-barrier mechanism for repository interoperability [OAI-PMH]
VC	Verifiable Credential	W3C standard for tamper-evident credentials [VCDM]
VP	Verifiable Presentation	Package of one or more VCs for presentation to verifier [VCDM]

Appendix B: References

Reference ID	Description	Link
[GX.CES]	Conceptual explenation, technical documentation, and high-level specification of the Credential Event Service	https://gitlab.com/gaia-x/lab/credentials-events-service https://gaia-x.eu/gaia-x-and-catalogues/ https://docs.gaia-x.eu/technical-committee/architecture-document/24.04/gx_services/#gaia-x-credential-event-service-ces
[DID]	Decentralized Identifiers (DIDs) v1.0	https://www.w3.org/TR/did-1.0/
[FACIS.FCE.SRS]	SRS for the Federation Architecture for Composed Infrastructure Services (FACIS) Enhancement of XFSC	https://github.com/eclipse-xfsc/docs/tree/main/federated-catalogue

	Federated Catalogue	
[FC.CCF.SRS]	SRS for the GXFS Federated Catalogue – Core Catalogue Features (FC.CCF)	https://www.gxfs.eu/download/1740/
[OAI-PMH]	The Open Archives Initiative Protocol for Metadata Harvesting	https://www.openarchives.org/OAI/openarchivesprotocol.html
[RFC.9727]	api-catalogue: A Well-Known URI and Link Relation to Help Discovery of APIs	https://datatracker.ietf.org/doc/rfc9727/
[SPARQL]	W3C SPARQL 1.2 Query Language	https://www.w3.org/TR/sparql12-query/
[VCDM]	Verifiable Credentials Data Model v2.0	https://www.w3.org/TR/vc-data-model-2.0/

Appendix C: Possible Implementation of the Local Catalogue with the XFSC Federated Catalogue

The following table explains, for the relevant requirements affecting the local catalogue, how they MAY be realized using the XFSC Federated Catalogue [FC.CCF.SRS, FACIS.FCE.SRS].

Functional Requirement	Related XFSC Federated Catalogue Requirement(s)	Comment
FR-ACM-02	(unspecified but implemented)	Regarding harvests performed by executing queries in a remote catalogue (case (3.) of FR-ACM-02), the XFSC Federated Catalogue has an (unspecified) feature for passing on incoming queries to “partner catalogues”. In contrast to the regular POST

		/query method, the */query/search* endpoint runs a query not only in the internal graph database, but also sends it to configured "partner" catalogues: • API definition: https://github.com/eclipse-xfsc/federated-catalogue/blob/main/openapi/fc_openapi.yaml#L713 • Implementation: https://github.com/eclipse-xfsc/federated-catalogue/blob/main/fc-service-server/src/main/java/eu/xfsc/fc/server/service/QueryService.java#L148 • Test: https://github.com/eclipse-xfsc/federated-catalogue/blob/main/fc-service-server/src/test/java/eu/xfsc/fc/server/controller/DistributedQueryControllerTest.java
FR-ACM-04	CAT-FR-SF-03, CAT-FR-LM-04	The XFSC Federated Catalogue is prepared to store, aside any machine-readable asset, a human-readable representation, which is subject to the same lifecycle. This functionality MAY be extended to link the original form of an imported asset to its transformed form after schema mapping.
FR-SR-01	CAT-FR-SF-02, -04, CAT-FR-CO-05	The XFSC Federated Catalogue supports schema management including versioning. The validity of assets against schemas is not enforced; instead, the user decides what schemas assets should be validated again. Still, additional modifications will be required to store remote schemas separately from local schemas.
FR-SR-09	CAT-FR-CO-02, -05	Using the XFSC Federated Catalogue as the local catalogue immediately satisfies the requirements for storing the result of validating an asset after schema mapping.
FR-CR-02	SD-Sch-01 SD-G-F-*	The XFSC Federated Catalogue features a schema storage and a graph database. Thus, it is possible to maintain the asset types as a dedicated ontology, or as a special named graph in the graph database, whichever suits better.

Appendix D: Change Log Template

For future revisions, document changes using this format:

Version	Date	Section	Change Description
[x.y]	[YYYY-MM-DD]	[Section number/name]	[Brief description of change]

--- End of Document ---