Software Requirements Specification

for

Gaia-X Federation Services

Federated Catalogue
Core Catalogue Features
FC.CCF

Published by

eco – Association of the Internet Industry (eco – Verband der Internetwirtschaft e.V.)

Lichtstrasse 43h

50825 Cologne

Germany

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Document Purpose

This document describes the requirements, external interfaces, and subsequent major design decisions for the Gaia-X Federation Service “Federated Catalogue”. This document contains the specification for a single instance of a Catalogue in an overall Federated Catalogue system.

To get general information regarding Gaia-X and the Gaia-X Federation Services please refer to [1] and [11].

1.2 Product Scope

The product scope is on the development of the Gaia-X Catalogue core features. The Gaia-X Catalogue makes Self-Descriptions of Providers and their offerings available to end-users and allows advanced queries for them.

Gaia-X Self-Descriptions express characteristics of Assets, Resources, Service Offerings and Participants. Self-Descriptions are tied to the identifier of the respective Asset, Resource or Participant. Providers are responsible for the creation of their Assets or Resources Self-Description. In addition to self-declared Claims made by Participants about themselves or about the Service Offering provided by them, a Self-Description may comprise Credentials issued and signed by trusted parties. Such Credentials include Claims about the Provider or Asset/Resource, which have been asserted by the issuer.

Self-Descriptions intended for public usage can be published into a Catalogue where they can be found by potential Consumers. The Providers decide in a self-sovereign manner which information they want to make public in a Catalogue and which information they only want to share privately. The goal of the (system of Federated) Catalogues is to enable Consumers to find best-matching offerings and to monitor for relevant changes of the offerings.

1.3 Definitions, Acronyms and Abbreviations

A more general Glossary about terms used within the Gaia-X context can be found in [1].

Term	**Definition**
Self-Description	File in the JSON-LD format use to describe a Participant, Asset or Resource in Gaia-X
Self-Description Graph	A graph database stores Self-Descriptions which cross-reference between each other
Object	Resources and Assets with a Self-Description managed by the catalogue

Gaia-X Federator	Federators are in charge of the Federation Services and the Federation which are autonomous of each other. Federators are Gaia-X Participants. There can be one or more Federators per type of Federation Service

Table 1: *Terms*

Abbreviation	**Definition**
AISBL	Association international sans but lucratif – the legal form of the non-profit organization of the Gaia-X Foundation
JSON-LD	JavaScript Object Notation – Linked Data
OWL	Web Ontology Language
RDF	Resource Description Framework
SD	Self-Description
SHACL	Shapes Constraint Language
SSI	Self-Sovereign Identity
VC	Verifiable Credential

Table 2*: Abbreviations*

1.4 References

[1] Gaia-X European Association for Data and Cloud, AISBL (2021): Gaia-X Architecture Document

Please refer to annex "Gaia-X_Architecture_Document_2103"

[2] JSON-LD 1.1. A JSON-based Serialization for Linked Data. W3C Recommendation 16 July 2020. https://www.w3.org/TR/2020/REC-json-ld11-20200716/

[3] Shapes Constraint Language (SHACL). W3C Recommendation 20 July 2017. https://www.w3.org/TR/2017/REC-shacl-20170720/

[4] openCypher: Query Language for Property Graphs. http://www.opencypher.org/

[5] Verifiable Credentials Data Model 1.0. Expressing verifiable information on the Web. W3C Recommendation 19 November 2019. https://www.w3.org/TR/2019/REC-vc-data-model-20191119/

[6] Specification of non-functional Requirements Security and Privacy by Design for Gaia-X Federation Services.
Please refer to annex "GXFS Nonfunctional_Requirements_SPBD"

[7] Data Catalog Vocabulary (DCAT) - Version 2. W3C Recommendation 04 February 2020. https://www.w3.org/TR/2020/REC-vocab-dcat-2-20200204/

[8] OpenID Connect Core 1.0, OpenID Foundation, 2014.
https://openid.net/specs/openid-connect-core-1_0.html

[9] OWL 2 Web Ontology Language. Structural Specification and Functional-Style Syntax (Second Edition). W3C Recommendation 11 December 2012. http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

[10] SKOS Simple Knowledge Organization System. W3C Recommendation 18 August 2009. http://www.w3.org/TR/2009/REC-skos-reference-20090818/

[11] Gaia-X European Association for Data and Cloud, AISBL (2021): Gaia-X Policy Rules Document
Please refer to annex "Gaia-X_Policy Rules_Document_2104"

[12] ADR Overview: Please refer to appendix D

[13] Gaia-X Federation Services Technical Development Requirements
Please refer to annex "GXFS_Technical_Development_Requirements"

[14] Gaia-X WP1 (2021), Architecture Overview
Please refer to annex "GX_IDM_AO"

2. Product Overview

2.1 Product Perspective

A Catalogue contains two types of storage: The Self-Description Storage for the published Self-Description files and the Self-Description Graph. By following references between Self-Descriptions in the graph, advanced queries across individual Self-Descriptions become possible.

Since Self-Descriptions are protected by cryptographic signatures, they are immutable and cannot be changed once published. The life cycle state of a Self-Description is described in additional metadata. There are four possible states for the Self-Description life cycle: “active” (the default), “revoked”, “deprecate” and “end-of-life”. The Catalogues provide access to the raw Self-Descriptions that are currently loaded including the life cycle metadata. This allows Consumers to verify the Self-Descriptions and the cryptographic proofs contained in them in a self-sovereign manner.

The Self-Description Graph contains the information imported from the Self-Descriptions that are known to the Catalogue and in an “active” life cycle state. The Self-Description Graph allows for complex queries across Self-Descriptions.

To present search results objectively and without discrimination, compliant Catalogues use a graph query language with no internal ranking of results: Besides the user-defined query statements with explicit filter and sort criteria, results are ordered randomly. The random seed for the search results can be set on a per-session basis so that the query results are repeatable within a session with the Catalogue.

In a privately hosted Catalogue, the authentication information can be used to allow a user to upload new Self-Descriptions and/or change the life cycle state of existing Self-Descriptions. In a public Catalogue, the cryptographic signatures of the Self-Descriptions are checked to make sure that the issuer of the Self-Description is the owner of the subject of the Self-Description. If that is the case, the Self-Description is accepted by the Catalogue. Hence, Self-Descriptions can be communicated to the Catalogue by third parties, as the trust verification is independent from the distribution mechanism. Self-Descriptions can be marked by the issuer as “non-public” to prevent that they are copied to a public Catalogue by a third-party that received the Self-Description file over a private channel.

A Visitor is an anonymous user accessing a Catalogue without a known account for the session. Every non-Visitor user interacts with the Catalogue REST API in the context of a session. Another option to interact with the Catalogue is to use a GUI frontend (e.g., a Gaia-X Portal or a custom GUI implementation) that uses the Catalogue REST API in the background. The interaction between the Catalogue and a GUI frontend is based on an authenticated session for the individual user of the GUI frontend.

A Provider can send Self-Descriptions about its offers to a Federated Catalogue, either via the remote API, which addresses machine clients, or using the portal, which addresses human end users. The Federated Catalogue stores and indexes the Self-Descriptions into a Self-Description Graph and makes it available to other Gaia-X Participants and Visitors to query for them.

The files containing the Self-Descriptions are stored in the Catalogue together with additional metadata. From these files, the SDs can be imported at any time into the Self-Description Graph, for instance, to recreate the graph and synchronize its content with other Catalogue instances. Consequently, only the set of combined Self-Description files is the ground truth for all other operations. Additional metadata, for instance, providing the life cycle state of the SD file, is further required.

2.2 Product Functions

2.2.1 User-Facing Functions

The main interaction with the Catalogue for an end-user is via the provided REST API. See Section 4.5 and Appendix B for more details on the API specification.

Note that the permissions below are intended for a standalone deployment of the Catalogue, such as in the context of a private company. In combination with a Self-Sovereign Identity system, the management of the access rights is done by the Gaia-X Participants in a self-sovereign manner. Furthermore, changing the life cycle of the Self-Descriptions depends on update messages with the signature of the issuer of the Self-Description.

In Table 3 below, a matrix of roles and their permissions on activities are shown. Each role and activity has an unique identifier. For activities starting with "Act" and for roles starting with "Ro".

The cells are filled with a "Yes", "No" or "Own". A "Yes" means that a user with this role has the permission to perform the activity on the entire scope of the Catalogue. For instance, the role "Catalogue Administrator" can read all participants in the Catalogue. A "Own" means that a user with this role can perform the activity only on the scope of the associated participant. A "No" indicates that there is no permission to perform the activity at all. A "*" on an activity description means that it is an internal activity with no associated API call.

The "Visitor" user is not mentioned in the table. This is the anonymous user that can perform only the activities that are always enabled ("Yes" for every user mentioned in the table).

		Internal Roles	**External Roles**		
		Ro-MU-CA	**Ro-MU-A**	**Ro-SD-A**	**Ro-Pa-A**
		Catalogue Administrator	Participant Administrator	Self-Description Administrator	Participant User Administrator
Participant					
Act-Pa-00	Add Participant	Yes	No	No	No
Act-Pa-01	Read Participant	Yes	Own	Own	Own
Act-Pa-02	Update Participant	Yes	Own	No	No
Act-Pa-03	Delete Participant	Yes	Own	No	No
Act-Pa-04	Get users of participant	Yes	Own	No	Own
User (For standalone deployment of a Catalogue without an existing IAM system)					
Act-Us-00	Create User	Yes	Own	No	Own
Act-Us-01	Read User	Yes	Own	No	Own
Act-Us-02	Update User	Yes	Own	No	Own
Act-Us-03	Delete User	Yes	Own	No	Own
Self-Description					
Act-SD-00	Add Self-Description	Yes	Own	Own	No
Act-SD-01	Get Self-Description	Yes	Yes	Yes	Yes
Act-SD-02	Update Self-Description	Yes	Own	Own	No
Act-SD-03	Revoke Self-Description	Yes	Own	Own	No
Act-SD-04	Verify SD trust	Yes	Yes	Yes	Yes
Act-SD-05	Verify SD syntax	Yes	Yes	Yes	Yes
Act-SD-06	*Inter Catalogue Synchronization Import	Yes	No	No	No

Act-SD-07	*Inter Catalogue Synchronization Export	Yes	No	No	No
Query					
Act-Qu-04	Execute a Query	Yes	Yes	Yes	Yes
Schema					
Act-Sc-00	*Add schema to schema storage	Yes	No	No	No
Act-Sc-01	Get the latest schema	Yes	Yes	Yes	Yes
Act-Sc-02	*Update schema	Yes	No	No	No
Act-Sc-03	*Delete schema	Yes	No	No	No
Act-Sc-04	*Verify schema	Yes	No	No	No
Act-Sc-05	*Import schemas	Yes	No	No	No
Act-Sc-06	*Export latest schemas	Yes	No	No	No
Role (For standalone deployment of a Catalogue without an existing IAM system)					
Act-Ro-00	Assign / Revoke public roles	Yes	Own	No	Own
Act-Ro-01	Assign / Revoke internal roles	Yes	No	No	No
Act-Ro-02	Get roles of a user	Yes	Own	No	Own
Act-Ro-03	Get list of all possible roles	Yes	Yes	Yes	Yes

***Table 3**: Role Activity Matrix*

Participant

The participant is described by a Self-Description and is the owner of all its Objects in the Catalogue. After a successful registration the participant gets an initial user account with all permissions on its own scope. The participant has several users underneath. For further information about the user and role structure, please have a look at Section 2.4.

User

A user belongs to one participant and can own roles. For standalone deployment of a Catalogue without an existing IAM system, users of a Catalogue instance are managed by the Catalogue itself. For further information about the user and role structure, please have a look at Section 2.4.

Self-Description

Every item in the Catalogue is described by a Self-Description. The Self-Description Administrator role does not include the permissions on the Self-Descriptions of Participants and Users, because their own roles are specified for them. Reading Self-Descriptions is allowed for each registered user and as well for non-registered users (Visitors). Section 4.1 specifies how Self-Descriptions are stored in the Catalogue.

Verification of Self-Descriptions is divided in two parts and is allowed for each registered user and as well for non-registered users:

a) Trust – Checking the signatures of the Self-Description
b) Syntax – Checking the syntactical correctness of the Self-Description against a Schema

For further information about the verification process, please have a look at Section 4.3.

Import and Export of Self-Descriptions aims at the Inter-Catalogue Synchronization (a different lot in the GXFS tender) and is an internal activity without an associated API call.

Query

In the first version of the Catalogue every registered user as well non-registered user are allowed to execute self-defined queries on the Self-Description Graph of the Catalogue. In future versions of the Catalogue, the options to manage self-defined queries can be expanded.

For further information about the structure and format of the queries, please have a look at Section 4.4.

Schema

Schemas define the structure of the Self-Descriptions. Each registered user and as well non-registered users can access the recent versions of all schemas. In the first version of the Catalogue, all other activities for the schema management are internal and not publicly available.

For further information about the schema management, please have a look at Section 4.2.

Role

For standalone deployment of a Catalogue, roles are managed by the Catalogue itself. For further information about the role structure and the provided roles, please have a look at Section 2.4.

2.2.2 Catalogue Management

The operator of the Catalogue needs to have dedicated administrative access: first to perform activities for which no dedicated API is implemented, secondly to monitor the status and resource consumption of the Catalogue.

2.2.3 High-Level Architecture

The following Figure 1 shows the high-level architecture with functional modules of the Catalogue. The next section introduces each module briefly. The detailed specification follows in Section 4 with the section numbers indicated in the high-level architecture.

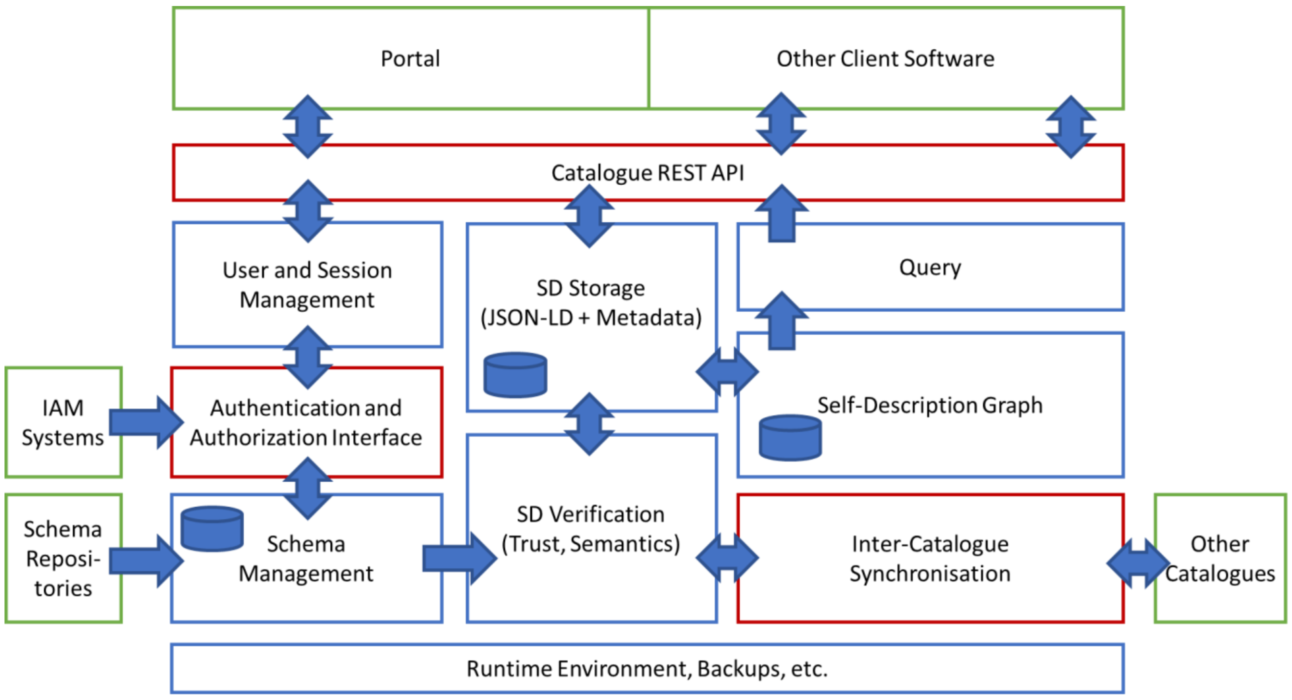

Portal
Other Client Software
Catalogue REST API
User and Session Management
SD Storage (JSON-LD + Metadata)
Query
Self-Description Graph
IAM Systems
Authentication and Authorization Interface
Schema Reposi-tories
Schema Management
SD Verification (Trust, Semantics)
Inter-Catalogue Synchronisation
Other Catalogues
Runtime Environment, Backups, etc.

***Figure 1**: High-Level Architecture of the Gaia-X Catalogue.*

Modules in blue provide internal functionality. Red modules indicate an external interface for the Catalogue. Green modules denote (potential) users of the external interfaces

Self-Description Storage
The Self-Description Storage subsystem holds the raw Self-Descriptions in JSON-LD format, as well as metadata information for the life cycle. The metadata cannot be stored in the raw JSON-LD format itself, because the JSON-LD part is protected by a cryptographic signature and cannot be changed. The Self-Description Storage subsystem is specified in Section 4.1.

Schema Management
Every Self-Description has to adhere to a schema definition. This ensures that information is structured in a uniform manner and queries can be formulated by the end-users. The Schema Management subsystem stores current and past versions of the schemas that are relevant to the Catalogue. The Schema-Management subsystem is specified in Section 4.2.

Self-Description Verification
When a Self-Description is uploaded to the Catalogue, it is verified for syntactic and trust conformance before being added to the Self-Description Storage. The Self-Description Verification subsystem is specified in Section 4.3.

Self-Description Graph
Self-Descriptions may cross-reference each other. The Catalogue queries shall allow following these references. For efficient processing of the queries, the Self-Descriptions with an “active” life cycle state are loaded into a graph database. The Self-Description Graph subsystem is specified in Section 4.4.

User and Session Management
The Catalogue software can be operated in conjunction with a Self-Sovereign Identity (SSI) system or standalone, for example with additional private Self-Description information in the context of a federation. In such a setting, the Catalogue has to manage its own participants.

Catalogue REST API
The REST API is the main external interface for users to interact with the Catalogue. The Catalogue REST API subsystem is specified in Section 4.5.

2.3 Product Constraints

This section summarizes the major constraints for the Catalogue. Many of them stem from definitions in Gaia-X ADRs (Architecture Decision Records). Please refer to the GXFS relevant ADRs in [12].

Self-Description Format
SDs are transferred using the JSON-LD format with signatures according to the Verifiable Credentials Data Model to prove their source and that they have not been intercepted during the communication or in the SD storage. All entities, both nodes and relations (i.e., edge labels), must have a URI as their unambiguous identifiers.

While this convention is closely aligned with the principles of Linked Data, entities of an SD Graph are not necessarily dereferenceable. In particular, a requesting client must not expect that it can find additional information at the Web resource identified by a URI used in the graph. Still, it is regarded as a good practice to supply such information and each data provider is encouraged to do so and comply with the Linked Data principles[1].

Self-Description Life cycle

As defined in ADR-003 [12], a Self-Description is in either of the following four states:

1. Active (the default state)
2. End-of-Life (after a timeout date, e.g., the expiry of a cryptographic signature)
3. Deprecated (by a newer Self-Description)
4. Revoked (by the original issuer or a trusted party, e.g., because it contained wrong or fraudulent information)

The default state is "active". The other states are terminal, i.e., no further state transitions follow upon them.

Note that the state of the Self-Description is independent from the state of the underlying entity. For example, a Service Offering can be deprecated (e.g., replaced by a newer version), whereas the Self-Description for the Service Offering (in that version) is still active.

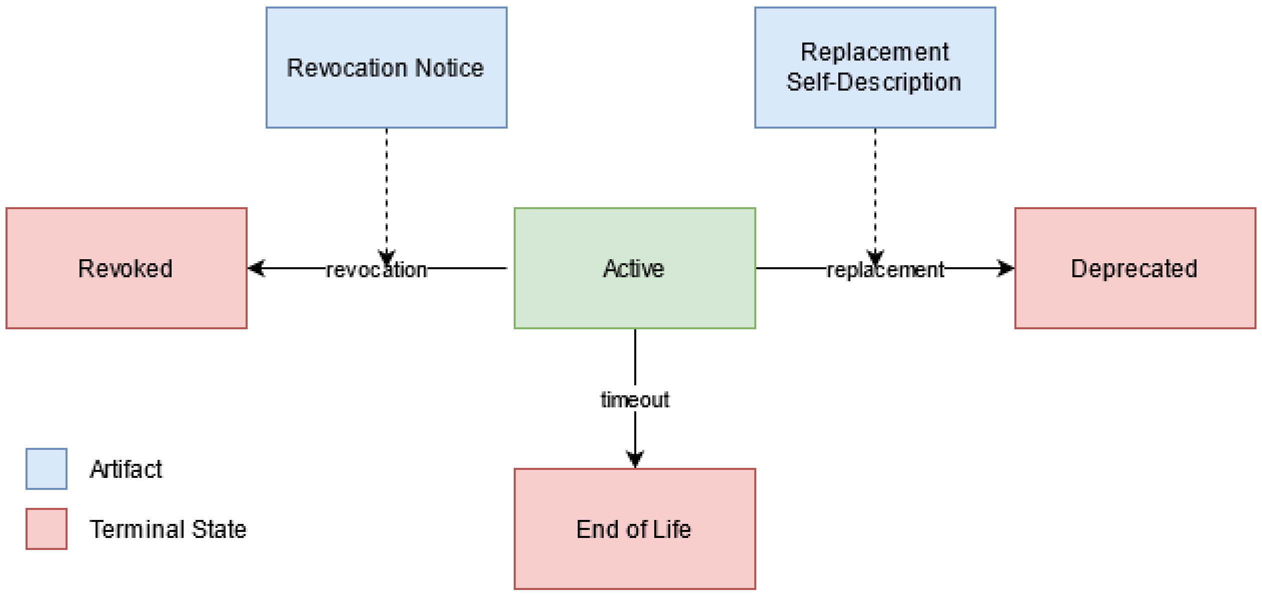


***Figure 2**: Self-Description Lifecycle*

Self-Descriptions are Self-Issued

The providers issue the Self-Descriptions in the form of Verifiable Presentations according to the Verifiable Credentials Data Model. The providers self-sign them with their publicly available cryptographic key (contained in the public Self-Description of the respective Provider) to avoid "fakes".

Frequent Renewal of Self-Descriptions

Similar to TLS encryption certificates that require frequent renewal, the Catalogue is "self-cleaning" by giving a limited timeout to all Self-Descriptions. No exact timeout has been defined so far. A consensus seems possible for a timeout of 90 days.

Query and Filter Functions

A Catalogue may provide support for several query languages, for instance openCypher, SPARQL or its

[1] https://www.w3.org/DesignIssues/LinkedData – not an official W3C Recommendation, but widely adopted

extension to property graphs SPARQL-star[2]. However, only openCypher must be supported by each Catalogue.

A Catalogue is not obliged to answer all incoming queries. Depending on actual or expected runtime but also if recognizing unsafe clauses as well as incoming requests with invalid or outdated authentication or authorization claims, then the Catalogue can reject the processing.

Federation of Catalogues

Services provided in the Gaia-X system should not be managed by a single party in the system, to avoid the possibility of a discriminating behavior of the Catalogue provider.

The onboarding to the Catalogue shall be possible in a non-discriminating way for all participants of Gaia-X. The participants must have the opportunity to describe, publish, maintain, and manage their description and different versions of their description in a self-sovereign way, so the issuer of the self-descriptions must be the providing participant itself. To ensure an easy onboarding of a provider the data and information schemas shall follow open standards.

Every federation shall be able to run his own Catalogue in his infrastructure to also be able to have private Catalogue entries for participant-internal services, which are not visible/exposed to external parties.

The Catalogue shall hold copies of the Self-Description files of the Providers of Assets, while the Catalogues shall also be able to handle updates of the Self-Description in a local cache. The local Self-Description Storage of a catalogue shall check the temporal validity of certified descriptions of the Assets or their Providers.

A mechanism for Catalogue Object Verification is currently considered based on DLT technologies. Not in scope for Release 1.

2.4 User Classes and Characteristics

In Gaia-X, each participant can manage their data themselves to guarantee data sovereignty. Thus, the participants need to interact with the Catalogue to manage their self-descriptions. Depending on the type of the participant, he has to do different tasks within Gaia-X and the Catalogue. The fulfillment of the tasks requires access rights to carry out actions within Gaia-X and the Catalogue. To assign different access rights to participants, a classification of the participants is required.

This chapter defines in a first step the structure of the user and role of participants interacting with the Catalogue in general to encourage a clear structural understanding of user classes and their characteristics. In a second step concrete roles are defined.

There are four key objects to describe the user management in the Catalogue:

1. Participants
2. User
3. Role
4. Activity

[2] https://w3c.github.io/rdf-star/

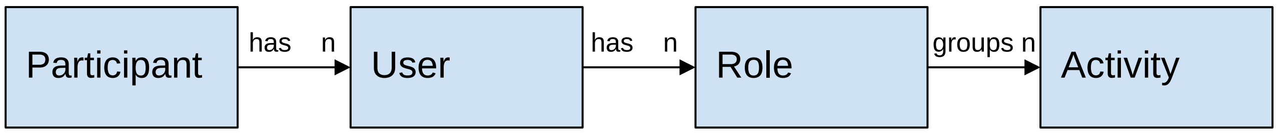


Figure 3:*User and Role Structure*

A participant can include users which are part of its organization. A user is assigned to roles, which give him the rights to perform actions in the Catalogue via the REST API (see Section 4.5). Thus, a role groups permission to perform some activities.

Participant
From the Gaia-X Architecture [1], a Participant is an entity, as defined in ISO/IEC 24760-1 as an *"item relevant for the purpose of operation of a domain [...] that has recognizably distinct existence",* which is onboarded and has a Gaia-X Self-Description.

A participant can be a provider of services and data or a consumer, which uses the provided services/data. Of course, a participant could have both roles at the same time. Each participant has to be registered in Gaia-X and is issued a certificate which is needed for identification within Gaia-X. The certification information for participants is managed by the IAM services by WP1[3] and issued in the form of Verifiable Credentials.

A special participant is the Gaia-X Federator for administration of infrastructure, internal services, and processes. For instance, a Trusted Signing Party will be announced and certified by a Gaia-X Federator. In private hosted Catalogues, the owner of the Catalogue is the Gaia-X Federator and has its permissions to the Catalogue instance.

A participant is assigned a unique identifier during his verification within the Gaia-X ecosystem. This unambiguous identifier is also used within the Catalogue.

User
A participant can have multiple users who can use the Catalogue API. Only users can login to the Catalogue and use the API. A participant signs the Verifiable Credentials of his users so that each user can identify himself to the Catalogue as a member of the participant. If a legal person wants to act on behalf of several participants, he needs a separate user for each participant. So, a user from the perspective of the Catalogue belongs to one single participant.

Possible kinds of a user:

- Principal – from the Gaia-X Architecture: "*A Principal is either a natural person or a digital instance who acts on behalf of a Gaia-X Participant*"
- Internal Users – e.g., technical users like a Catalogue administrator. Users for maintenance and administrative tasks with expanded permissions within the Catalogue.
- Visitor – from the Gaia-X Architecture: "*Anonymous, non-registered entity (natural person, bot, ...) browsing a Gaia-X Catalogue*"

To enable access right management for user actions with the Catalogue API, there is a need to define roles, which a user can be assigned to. A user has specific roles that authorize him to execute certain API calls.

[3] Please refer to appendix E for an overview and explanation of the Work Packages (WP).

A user has an identifier, which is unique in the Catalogue instance and in the scope of the associated participant. The identifier could be, for instance, a composition of the username and the associated participant identifier.

Role

A role is defined by a group of activities, in particular the permissions to perform these contained activities. The Catalogue provides several roles, which can be assigned to a user (see table 4).

To provide meaningful roles, the following objects of the activities in Section 4.5 are considered:

- Participant
- User
- Self-Description (for, e.g., provided Assets)
- Query
- Schema Management
- Role

These objects are managed in the Catalogue. The users have to do Create, Read, Update and Delete (CRUD) operations with most of the objects.

The Catalogue checks the permissions of the user when he creates a session. The permissions of the user are valid during the whole session. After a configurable timeout, the user has to create a new session (e.g., after 2 days). The timeout can be configured within the Catalogue.

Possible roles will be provided by the Catalogue and can be used by the participants to assign to their users. Therefore, the Catalogue stores the roles in a database and provides them via the REST API (see Section 4.5. In the first version of the Catalogue a Role Based Access Management is provided to keep it as simple as possible.

The Catalogue uses OpenID Connect on top of OAuth2 for authentication of a user. For details, please have a look in the specification document “IDM & Trust – Architecture Overview” by WP1 [14].

Note that the role-based access management below is intended for a standalone deployment of the Catalogue. Such as in the context of a private company. In a Gaia-X hosted Catalogue the access management in combination with a Self-Sovereign Identity system is performed by the Gaia-X Participants in a self-sovereign manner.

Role Assignment

A role is stored as a Verifiable Credentials (VC) of a user. The VC is signed by the participant to confirm the validity of the content. If an additional role is assigned to a user, the participant is to create and sign a new VC of the user with the newly added role.

Roles determine the authorization of actions on objects. A participant needs the opportunity to assign roles to its users. Therefore, there is a need for roles with appropriate permissions. There are two types of roles in general:

1. External roles – publicly available roles. Every participant can assign these roles to its users.
2. Internal roles – special roles with huge sets of permissions. Only special participants can assign these roles to users. Used for internal activities like maintenance and administration of the Catalogue.

Hence, it is necessary to distinguish between the role assignment for these two types of roles. External roles can be assigned by any user of a participant with the appropriate roles for role assignment. Otherwise, internal roles can only be assigned by users of a Gaia-X Federator with the appropriate roles.

Initial Assignment and Registration of a Participant in the Catalogue

The initial onboarding of a Participant is defined in a separate lot of the GXFS tender "Onboarding & Accreditation Workflows".

After a successful registration of a participant in the Catalogue instance, a master user is automatically created and assigned to the participant. With this master user the participant has permission rights to all actions to its own Objects (e.g., all Self-Descriptions of the participant). One of the first actions of the master user could be to create users in his organization with appropriate roles.

Registration process:

1. A Participant registers in the Catalogue through the API, e.g., via a platform like the Portal specified in Work Package 5[4].
2. The Catalogue verifies the provided signatures of the participant with an external Authority like the AISBL.
3. The Catalogue requests the Self-Description of the Participant verified by the Onboarding & Accreditation Workflow of Work Package 4[5].
4. The Catalogue confirms the registration by adding the Self-Description of the participant to its database and creating of an initial user account for the participant with the two main intentions:
    a. The initial user has all permissions in the context/scope of his organization. So, the user can perform all external activities on Objects of the associated participant in the Catalogue.
    b. The initial user can add more users and assign roles to them in the context of his organization.

Permission check

The role information of a user is stored in the Verifiable Credentials (VC) and confirmed by the participant via a signed certificate. The Catalogue gets the VC out of the OAuth2/OpenID connect token by sending the token to an external service (called SSI Extension Shell) provided by WP1. For details, please have a look in the specification document "IDM & Trust – Architecture Overview" by WP1 [14].

The Catalogue compares the needed permissions to perform the requested action and the permissions of the requesting user. If the user has the needed permissions the Catalogue allows the request, otherwise the Catalogue forbids the request.

Permission check process:

1. User logs in to the portal (or another platform which gains an access token) with his user credentials
2. User sends an API request to the Catalogue via the portal with an OAuth2/OpenID Connect access token
3. Catalogue forwards the access token to an external service called SSI Extension Shell provided by WP1

[4] Please refer to appendix E for an overview and explanation of the Work Packages (WP).
[5] Please refer to appendix E for an overview and explanation of the Work Packages (WP).

4. If the access token is valid, the SSI Extension Shell sends the associated verifiable credential (VC) of the user to the Catalogue
5. Catalogue resolves the role id(s) out of the VC
6. Catalogue checks the permission rights of the role
7. Catalogue compares the assigned permission rights with the needed permissions rights to perform the requested API call
8. Catalogue allows and executes the API call or forbids it. In case of rejection the Catalogue sends an error message with status code 403 (Forbidden)

Scope of a Role

The scope of a role describes the breadth of the authorization from the perspective of the participant. In the Catalogue there are several participants with their own users and Self-Descriptions. Each participant has the responsibility for its own Objects – for its own scope. For administrative purposes there are participants in the Catalogue which can manipulate all Objects of all registered participants. So not only for its own scope but for all scopes.

In particular there are two possible scopes:

a) Own – all Objects of the participant of which it is the owner from
b) All – all Objects in the Catalogue, regardless of the ownership of the Object

Ownership of Self-Descriptions

The owner of a Self-Description is indicated by the attribute *providedBy* in the associated JSON-LD file of the Self-Description. The value of this property is a reference to a participant which is designated as owner of the Self-Description from the perspective of the Catalogue.

The owner of a Self-Description from the perspective of the Catalogue could be another one as the legal owner of the described physical or virtual asset in the Self-Description. The Federated Catalogue assumes that the participant has the legal rights to add a Self-Description of the physical or virtual asset.

Defined Roles

The Catalogue must provide a set of roles which can be assigned to users. However, the configuration should be open for easy adaptation of the roles to be prepared for further versions of the Catalogue. There must be documentation for the administrators/software engineers of the Catalogue how to add new roles, adjust existing roles and delete roles.

Each role gets a unique identifier, which is used to reference the roles. In particular, the Verifiable Credential of the user contains a list of role identifiers of the roles that are assigned to the user.

The table below lists the four provided roles in the Catalogue. In the activity role matrix in section 2.2 exists an overview about all provided roles and their permissions based on the activities from section 4.5.

ID	Description	Permissions	Scope
Ro-MU-CA	Catalogue Administrator	All permissions to external and internal activities of all Objects in the Catalogue. The abbreviation MU derives from the term Master User.	All

Ro-MU-A	Participant Administrator	All permissions to external activities of the own scope of the participant. The abbreviation MU derives from the term Master User.	Own
Ro-SD-A	Self-Description Administrator	Users with this role are allowed to create, read, update, and delete the Self-Descriptions of the associated participant. E.g., the Self-Description of all services of the associated participant.	Own
Ro-Pa-A	Participant User Administrator	Users with this role are allowed to create, read, update, and delete users of the associated participant. Additionally, they are allowed to and revoke external roles to other users of the associated participant. This role is intended for the administration of users of the participant.	Own

***Table 4**:Provided Roles by the Catalogue*

2.5 Operating Environment

Please refer to [13] for further binding requirements regarding the operating environment.

2.6 User Documentation

The documentation artifacts are grouped into three main categories: Catalogue user documentation, operator documentation and developer documentation.

Catalogue User Documentation
The End-User Documentation comprises at least the following elements:

- Feature-Complete Description of the REST API
- Description of the openCypher query interface with examples

Catalogue Operator Documentation
The Operator Documentation comprises at least the following elements:

- Setup and configuration of a new Catalogue instance
- Description of operator-specific system access
  - Logging
  - Metrics collection
- Description of the maintenance workflows
  - Backups
  - Schema Management

Developer Documentation
The Developer Documentation comprises at least the following elements:

- All internal APIs are documented
  - Can be generated from source-code annotations if such are available for the selected programming language.
- The source code has to consist of at least 5% comments.

- Development history from a source code versioning system like git or Subversion.
- Commit messages with motivation for changes.

Please refer to [13] for further requirements regarding documentation.

2.7 Technical Guidance

The major functional dependencies are to the adjacent Gaia-X Federation Services.

The major dependencies on software packages from third parties are mitigated by the selection of technologies for which multiple Open-Source implementations exist. The following paragraphs denotes some software components where usually an existing software would be used.

- Webserver for the REST Interface
  - Apache2, Nginx
- Graph Database
  - Neo4J Community Edition, Redis Graph
- Self-Description Administrative Metadata Database
  - MongoDB, Neo4J Community Edition, MySQL/Postgres/SQLite
- Local User Management
  - OpenLDAP, Keystone, Keycloak
- Verifiable Credentials - Verification of Signatures
  - JSON Web Token: See list at https://openid.net/developers/jwt/
- Distributed Ledger Technology
  - ARIES, idUnion

3. Requirements

Further requirements can be found in [13].

3.1 External Interfaces

3.1.1 User Interfaces

The Catalogue provides no user interfaces. The Gaia-X Portal (a separate lot in the GXFS tender)provides a user interface on top of software interfaces by the Catalogue.

3.1.2 Hardware Interfaces

The Catalogue provides no hardware interfaces.

3.1.3 Software Interfaces

The Catalogue provides no “local” software interfaces. See the next section for the communication interfaces. The data structures used in externally exposed interfaces are the following:

1. Self-Descriptions (JSON-LD)
2. Self-Description Schemas (RDF Schema, OWL, SHACL)
3. Verifiable Credentials with user information (participant to which the user belongs, roles, etc.)
4. Self-Description Life Cycle State Update Messages (i.e., REST API calls)

Section 3.2.4 explains data structures (1.) to (3.) more specifically.

3.1.4 Communications Interfaces

As depicted in the High-Level Architecture, the Catalogue provides three main external communication interfaces.

- REST API
- Authentication and Authorization

All three are described in more detail in Section 4 on the System Features.

The communication with the Catalogue REST API is secured with TLS v1.2 or above.

3.2 Functional Requirements

This section describes functional requirements for the overall system. See Section 4 for detailed requirements for specific subsystems.

3.2.1 Logging

Some components in the Catalogue, such as databases, provide their own logging facilities. These are typically able to log into a common log aggregator like syslog, Log4J or similar. Logging across all subsystems has to be configurable so that an operator can route logs to a central (possibly external) log aggregator if he wishes to do so.

3.2.2 Backups

The Catalogue state has to be 100% recoverable from backups. An automated backup routine collects all relevant data and prepares a file-system folder (alternatively a tarball) with compressed data dumps. Backups of the full system must not interrupt services for more than 10 minutes in a scenario with 1 million Self-Descriptions. Recovering from backups in that scenario must not take longer than 60 minutes.

3.2.3 Scaling

Subsystems that can become bottlenecks in the number of served requests are implemented in such a way that enables parallelization. All operations that are accessible from the public API can scale to at least 100 requests per second if not stated otherwise in the detailed non-functional requirements.

3.2.4 Self-Description

The Self-Description documents must contain the metadata about a service offering of Resources, Assets and Participants in Gaia-X. In the Catalogue, Self-Descriptions must be presented as raw JSON-LD files. Furthermore, the published Self-Descriptions have to be protected by cryptographic signatures according to the Verifiable Credentials Data Model, so that they are immutable via communication. In technical terms, every Self-Description is a Verifiable Presentation of one or more Verifiable Credentials, which makes claims about the subject of the Self-Description. The RDF triples inside the Verifiable Credentials have to be valid instances of the Self-Description Schema.

See the following code listing for a complete demonstration of VCs embedded into a Verifiable Presentation. This example splits a Provider Self-Description into multiple Credentials and adds proofs from particular (external) institutions to the respective parts. In this example, the street address is signed by the German Handelsregister (company register), while ISO-9001 and DUNS declarations are each signed by respective (external) institutions. At the very end, the Provider themselves adds a final proof to the overall Verifiable Presentation of all Credentials.

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://w3id.org/gaia-x/context.jsonld"
  ],
  "type": "VerifiablePresentation",
  "verifiableCredential": [
    {
      "@context": [
        "https://www.w3.org/2018/credentials/v1",
        "https://w3id.org/gaia-x/context.jsonld"
      ],
      "id": "http://example.edu/credentials/1872",
      "type": "VerifiableCredential",
      "issuer": "https://www.handelsregister.de/",
      "issuanceDate": "2010-01-01T19:73:24Z",
      "credentialSubject": {
        "@id": "http://example.org/add-your-provider-id-or-website-here",
        "@type": "gax:Provider",
        "gax:hasLegallyBindingName": "My example provider",
        "gax:hasLegallyBindingAddress": {
          "id": "_:b0",
          "type": "vcard:Address",
          "vcard:street-address": "Example street 2",
          "vcard:postal-code": "99999",
          "vcard:country-name": "Country Name 2",
          "vcard:locality": "City Name 2"
        }
      },
      "proof": {
        "type": "RsaSignature2018",
        "created": "2017-06-18T21:19:10Z",
        "proofPurpose": "assertionMethod",
        "verificationMethod": "https://example.edu/issuers/keys/1",
        "jws":
"eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..TCYt5XsITJX1Cx
PCT8yAV-TVkIEq_PbChOMqsLfRoPsnsgw5WEuts01mq-pQy7UJiN5mgRxD-
WUcX16dUEMGlv50aqzpqh4Qktb3rk-BuQy72IFLOqV0G_zS245-
kronKb78cPN25DGlcTwLtjPAYuNzVBAh4vGHSrQyHUdBBPM"
      }
    },
    {
      "@context": [
        "https://www.w3.org/2018/credentials/v1",
        "https://w3id.org/gaia-x/context.jsonld"
      ],
      "id": "http://example.edu/credentials/1874",
      "type": "VerifiableCredential",
      "issuer": "https://www.example-cert-institution.de/",
      "issuanceDate": "2010-01-01T19:73:24Z",
      "credentialSubject": {
        "@id": "http://example.org/add-your-provider-id-or-website-here",
        "@type": "gax:Provider",
        "gax:hasCertification": "gax:ISO_9001"
      }
    },
    {
      "@context": [
        "https://www.w3.org/2018/credentials/v1",
        "https://w3id.org/gaia-x/context.jsonld"
      ],
      "id": "http://example.edu/credentials/1875",
      "type": "VerifiableCredential",
      "issuer": "https://www.example-identifier-institution.org",
      "issuanceDate": "2010-01-01T19:73:24Z",
      "credentialSubject": {
        "@id": "http://example.org/add-your-provider-id-or-website-here",
        "@type": "gax:Provider",
        "gax:hasBusinessIdentifier": {
          "@id": "_:b1",
          "@type": "gax:BusinessIdentifier",
          "gax:hasIdentifierNumber": "12-345-6789",
          "gax:hasIdentifierSystem": "DUNS"
        }
      }
    }
  ],
  "proof": {
    "type": "RsaSignature2018",
    "created": "2018-09-14T21:19:10Z",
    "proofPurpose": "authentication",
    "verificationMethod":  "did:example:ebfeb1f712ebc6f1c276e12ec21#keys-
1",
    "challenge": "1f44d55f-f161-4938-a659-f8026467f126",
    "domain": "4jt78h47fh47",
    "jws":
"eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..kTCYt5XsITJX1C
xPCT8yAV-TVIw5WEuts01mq-
pQy7UJiN5mgREEMGlv50aqzpqh4Qq_PbChOMqsLfRoPsnsgxD-WUcX16dUOqV0G_zS245-
kronKb78cPktb3rk-BuQy72IFLN25DYuNzVBAh4vGHSrQyHUGlcTwLtjPAnKb78"
  }
}
```

3.2.5 Performance Requirements

3.2.5.1 Response Time

Response times of various activities in a Catalogue are clearly identified. Measurement points include the response time of updating a Self-Description, the result response of a query request as well as synchronization requests of distributed Self-Descriptions. 95% of all response time should be less than 5 seconds. In case of longer delays, the Catalogue will give feedback indicating when it expects the operation to be done.

3.2.5.2 Workload

The Catalogue must support the following workload:

- Catalogue must handle 100 read transaction of Self-Description per second
- Catalogue must handle 50 update transactions of Self-Description per second
- Catalogue must handle 20 query transactions over the Self-Description Graph per second

3.2.5.3 Accuracy

The Catalogue must provide high accuracy for returned results of query requests. The accuracy of returned results must be higher than 99%.

3.2.6 Safety Requirements

None

3.2.7 Security Requirements

As one of the Gaia-X Federation Services, Federated Catalogue must ensure Security and Privacy by Design (SPBD) documented in [6]. In general, constructing Security and Privacy by design requirements is considered in two steps. The first step is to define protection goals of assets deployed on each technical component. Protection goals including confidentiality, integrity and non-repudiation for each asset are listed in [6] Table 2. Then, based on the deduction in the first procedure, the protection profile of individual technical components in the Catalogue can be derived (indicated in Table 3 in [6]), which leads to the classification of Assurance Levels for Catalogue. In conclusion, three technical components in Catalogue, which are Catalogue Query, Catalogue Management, and Inter-Catalogue Synchronization, own a basic Assurance Level. A general overview of standards related to security and privacy is provided in Table 7 of [6].

The EUCS defines 20 control categories [6]. Several of them are applicable to Gaia-X Federated Catalogue and the details of each requirement are listed as the following.

3.2.7.1 Identity, Authentication, and Access Control Management

ID	Description	Acceptance Criteria	Priority
SR-Access-01	Each client must have a correct identity claim.	Execute a request from a client with an arbitrary identity claim. This request must be rejected.	Mandatory (Phase 1)
SR-Access-02	Identity of a signing party of the identity claim is not revoked.	Execute a request from a client, whose identity claim is signed by a revoked identity. This request must be rejected.	Mandatory (Phase 1)
SR-Access-03	The Catalogue uses OpenID Connect on top of OAuth2 for authentication of a user.	Execute a request without an authorizationUrl, the request must be rejected.	Mandatory (Phase 1)
SR-Access-04	All elements in the Catalogue need to be protected from unauthorized access.	Execute a query request for protected information with: a) no user information (request must be rejected with response stating 'unauthorized'), b) arbitrary user information (request must be rejected with response stating 'unauthorized'), c) correct user information of a user who is not allowed to access the target information (request must be rejected with response stating 'unauthorized'), d) user that is allowed to access the information and with correct authentication information (request is processed and information returned)	Mandatory (Phase 1)
GraS-SR-01	Backup of Self-Description Graph needs to be encrypted.	The data encryption should be performed on the client before sending the copy to the server that will keep it.	Mandatory (Phase 1)

Table 5:*Security Requirements Identity, Authentication, and Access Control Management*

3.2.7.2 Cryptography and Key Management

ID	Description	Acceptance Criteria	Priority
SR-Crypto-01	All elements in the Catalogue must be protected by cryptographic signatures according to the Verifiable Credentials Data Model.	Execute a request to an arbitrary element in the Catalogue without a cryptographic signature, the request must be rejected.	Mandatory (Phase 1)
SR-Crypto-02	Query can ask for specific signatures or characteristics.	The Federated Catalog must provide the cryptographic proofs (signatures of the underlying JSON-LDs) to enable the querying system to check the correctness of the query result.	Mandatory (Phase 1)

SR-Crypto-03	A signature indicating the date of the incoming messages needs to be logged.	There must be timestamps for all incoming messages.	Mandatory (Phase 1)
SR-Crypto-04	A signature indicating the one-to-one relation between Self-Description and Self-Description Graph is necessary.	The proof signature of the SD in the SD Storage System must be equal to the related SD resource in the SD Graph.	Mandatory (Phase 1)
SR-Crypto-05	Express required trust of the Self-Descriptions in the query.	The sender of the query is able to set a flag to ask also for SDs with, for instance, expired or revoked states. If the flag is set, the returned result list must contain the respective matching SDs. If the flag is not set, only valid SDs with proper signatures and in the 'active' life cycle state must be returned.	Mandatory (Phase 1)
SR-Crypto-06	Backup of Self-Description Graph needs to be encrypted.	The import of a previously backed up SD Graph must only be possible if the correct password is used. The SD Graph must not export its content without setting an encryption password.	Mandatory (Phase 1)

Table 6*: Security Requirements Cryptography and Key Management*

3.2.7.3 Communication Security

ID	Description	Acceptance Criteria	Priority
SR-Comm-01	The outside communication with REST API needs to be encrypted.	Executing a query request without encryption or invalid encryption will be rejected.	Mandatory (Phase 1)
SR-Comm-02	Only TLS interaction is supported.	Only state-of-the-art TLS versions, i.e., > 1.2, should be in use.	Mandatory (Phase 1)
SR-Comm-03	The incoming messages must be valid and uncorrupted.	Send a corrupted message to a Gaia-X Federated Catalogue, it must reject it.	Mandatory (Phase 1)

Table 7*:Security Requirements Communication Security*

3.2.8 Data Standards

ID	Description	Acceptance Criteria	Priority
DS-01	The schemas used for the self-description SHALL follow open standards	Definition and/or usage of an open non-proprietary standard	Mandatory (Phase 1)

		available also for future participants	

***Table 8**:Data Standards*

4. System Features

The Gaia-X Catalogue follows a Micro-Service architecture with several different modules as stated in Section 2.2.4. The following requirements define these modules and describe their respective interfaces.

4.1 Self-Description Storage Subsystem

4.1.1 Description

Self-Descriptions are represented as graphs in the RDF data model, using terms from the Self-Description Schema. Gaia-X has adopted JSON-LD as its preferred serialization for RDF (cf. ADR-001 in [12]). The Federated Catalogue stores each Self-Description as one JSON-LD file, once it has accepted it as valid.

The Provider or Asset owner who submitted a Self-Description may update it. The Federated Catalogue will retain previous versions of such updated Self-Descriptions, and clients will be able to retrieve each version of a Self-Description.

For the purpose of querying, the most recent versions of all Self-Descriptions will be indexed into the Self-Description Graph as explained in Section 4.4.

To organize the indexing, versioning, etc. of Self-Descriptions, the Federated Catalogue maintains Administrative Metadata about each Self-Description, including in what state of the life cycle according to ADR-003 [12] it is, etc. These metadata are not included inside the Self-Descriptions and are not required to be expressed in terms of the Self-Description Schema. Instead, the schema of Administrative Metadata is specified informally, as follows[6]:

Key	**Datatype**	**Description**
id	URI	The identifier of the Participant or Asset being described, same as the common *credentialSubject* of the Verifiable Credentials inside the Self-Description
hash	string (hexadecimal)	A hashcode computed over the JSON-LD Self-Description
issued	date (xsd:dateTimeStamp)	Date when the Self-Description was originally issued by the Participant
received	date (xsd:dateTimeStamp)	Date when the Self-Description was first received by this Federated Catalogue

[6] See also Appendix C for a possible representation of the Self-Description metadata in SQL.

state	one of *"active"*, *"deprecated"*, *"revoked"* or *"eol"* (for "end-of-life")	State in the Self-Description life cycle as defined in Section 2.3
issuedBy	URI	Identifier of the Participant that has issued the Self-Description

***Table 9**: Self-Description Keys, Datatypes and Descriptions*

A system MAY use further keys.

Four kinds of REST API Requests to Self-Description Storage

There are four types of API requests to Federated Catalogue, each of which gets access to Self-Description Storage. The workflow of them is introduced as the following respectively and more details of REST API Requests are specified in Section 4.5.

The first kind of API request is for data consumers to retrieve a list of the Administrative Metadata of those SDs such that these Administrative Metadata match certain given filters, as indicated in Figure 4. A variant of this request takes a query for Participants or Assets instead of simple filters, executes the query over the SD Graph as specified in Section 4.4, but returns, rather than the query result as returned by the graph database, the Administrative Metadata of the SDs in the result set of the query.

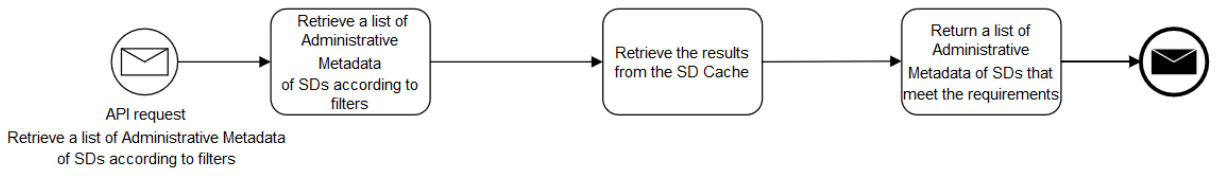

API request
Retrieve a list of Administrative Metadata of SDs according to filters
Retrieve a list of Administrative Metadata of SDs according to filters
Retrieve the results from the SD Cache
Return a list of Administrative Metadata of SDs that meet the requirements

***Figure 4**: Workflow of retrieving a list of Administrative Metadata of SDs based on filters*

Then, for data consumers, there is another variant of this request, which retrieves, in addition to the Administrative Metadata, also the raw JSON-LD Self-Descriptions. The process of this procedure is depicted in Figure 5.

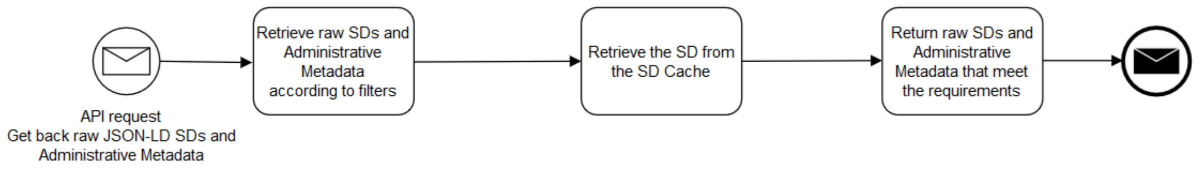

API request
Get back raw JSON-LD SDs and Administrative Metadata
Retrieve raw SDs and Administrative Metadata according to filters
Retrieve the SD from the SD Cache
Return raw SDs and Administrative Metadata that meet the requirements

***Figure 5**: Workflow of retrieving raw JSON-LDs and Administrative Metadata*

Technically, the result of such a query, which combines raw JSON-LD Self-Descriptions and Administrative Metadata shall have the following structure[7]:

```
{
    "results": [
```

[7] This is inspired by the result format of the Neo4j Cypher transaction API, cf. https://neo4j.com/docs/http-api/current/actions/result-format/. It is not required to refer to the JSON-LD context of the Verifiable Credentials Data Model exactly as given below, but the column type https://www.w3.org/2018/credentials#VerifiablePresentation must be used, and a reference to that context is a straightforward way of achieving that.

```
        {
            "@context": "https://www.w3.org/2018/credentials/v1",
            "columns": [ "VerifiablePresentation" ],
            "data": [
                {
                    // first matching Self-Description
                    "row":   [   {   "type":   "VerifiablePresentation",
"verifiableCredential": …, "proof": … } ],
                    "meta": [ { "issued": …, "received": …, "state": … }
]
                },
                {
                    // second matching Self-Description (if any)
                    "row":   [   {   "type":   "VerifiablePresentation",
"verifiableCredential": …, "proof": … } ],
                    "meta": [ { "issued": …, "received": …, "state": … }
]
                },
                …
            ]
        }
    ]
}
```

Listing 4.1: Result format for Raw JSON-LD Self-Descriptions with Administrative Metadata

Note that, in large federations, it may be necessary to distinguish Self-Descriptions by their application domain or the data space / data ecosystem they originated from, e.g., “healthcare” vs. “logistics”, and to retrieve only Self-Descriptions from one of these domains. This, however, does not require an additional Administrative Metadata field, but is supported because the Core Ontology, which provides the foundation for any Self-Description Schema, provides the *gax:theme* property, a specialization of *dcat:theme* from DCAT, which indicates one or more categories that a resource belongs to. Any such category is identified by a URI, which should point to a concept in a Controlled Vocabulary. This may be an external, existing Controlled Vocabulary, or one that is maintained locally using the Schema Storage Subsystem of the Federated Catalogue (cf. Section 4.2).

When a new SD is added to Catalogue or the status of life cycle needs to be updated, the SHACL Module and Cryptographic signature validation workflows must be executed, which are represented in Figure 6 and Figure 7 respectively. Details of Cryptographic signature validation is in Figure 9 in Section 4.3.

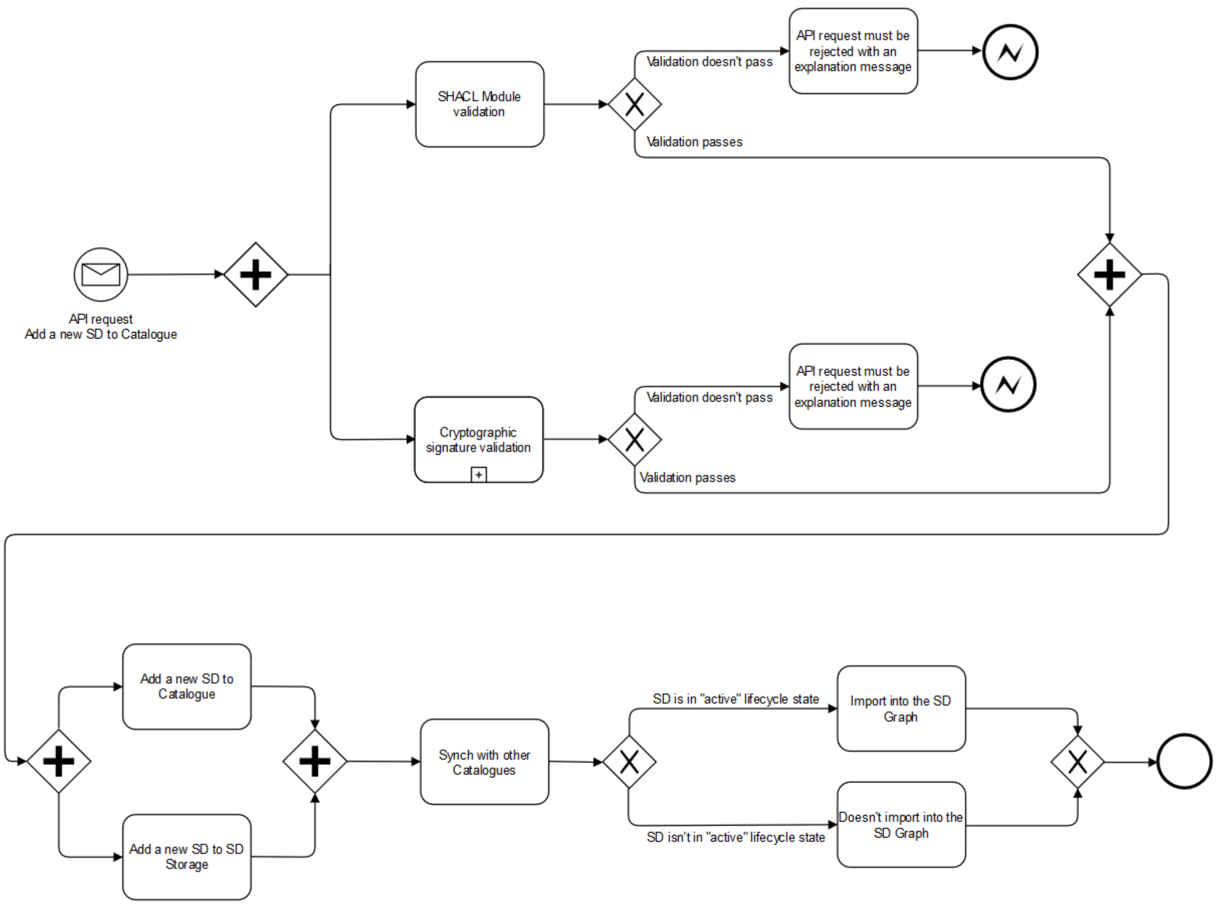

API request
Add a new SD to Catalogue
SHACL Module validation
Validation doesn't pass
API request must be rejected with an explanation message
Validation passes
Cryptographic signature validation
Validation doesn't pass
API request must be rejected with an explanation message
Validation passes
Add a new SD to Catalogue
Add a new SD to SD Storage
Synch with other Catalogues
SD is in "active" lifecycle state
Import into the SD Graph
SD isn't in "active" lifecycle state
Doesn't import into the SD Graph

***Figure 6**:Workflow of adding a new SD to Catalogue*

The step "Synch with other Catalogues" is optional and handled by the future Inter-Catalogue Synchronization module according to Sync-F-06. Not in scope for Release 1.

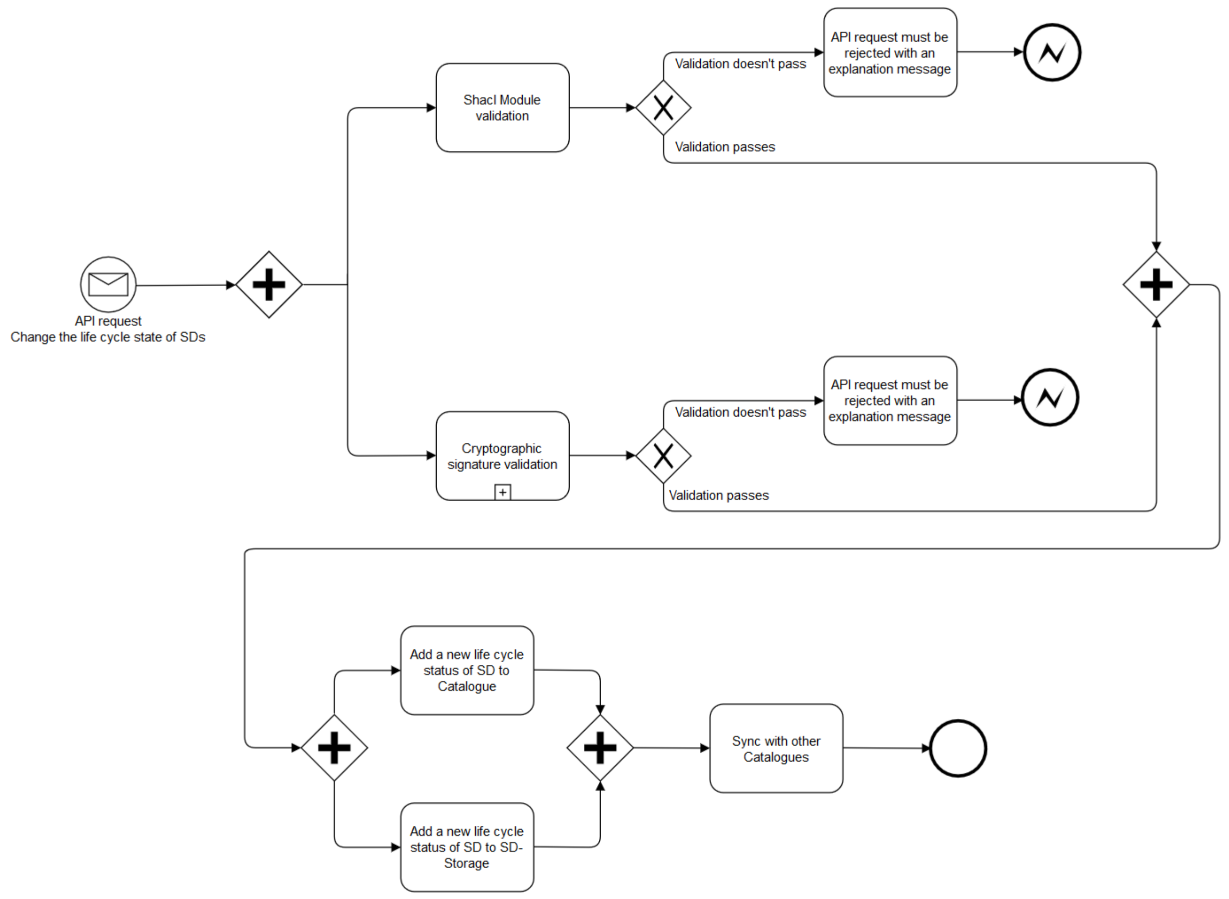

API request
Change the life cycle state of SDs
Shacl Module validation
Validation doesn't pass
API request must be rejected with an explanation message
Validation passes
Cryptographic signature validation
Validation doesn't pass
API request must be rejected with an explanation message
Validation passes
Add a new life cycle status of SD to Catalogue
Add a new life cycle status of SD to SD-Storage
Sync with other Catalogues

***Figure 7**:Workflow of changing the life cycle state of SDs*

4.1.2 Functions

4.1.2.1 Adding a new Self-Description

ID	Description	Acceptance Criteria	Priority
SD-S-S-01	A Provider MUST be able to add a new Self-Description to the Catalogue	"Workflow of adding a new SD to Catalogue" implemented according to Figure 6	Mandatory (Phase 1)
SD-S-S-02	The Federated Catalogue MUST be able to synchronize a new Self-Description from a remote Federated Catalogue.	Like SD-S-S-01, but the action is performed by the Inter-Catalogue Synchronization mechanism rather than by a Participant.	Mandatory (Phase 1)

***Table 10**: Self-Description Storage Subsystem Functions - Adding a new Self Descriptions*

4.1.2.2 Syntactic Validation

ID	Description	Acceptance Criteria	Priority
SD-S-V-01	The Federated Catalogue MUST be able to validate the syntactic well-formedness of Self-Descriptions against the JSON-LD grammar[8].	Unit test with a well-formed and a non-well-formed JSON-LD input	Mandatory (Phase 1)

[8] https://www.w3.org/TR/2020/REC-json-ld11-20200716/#json-ld-grammar

Table 11: *Self-Description Storage Subsystem Functions - Syntactic Validation.*

4.1.2.3 Changing the Life Cycle State of Self-Descriptions

ID	Description	Acceptance Criteria	Priority
SD-S-U-01	The Federated Catalogue MUST accept requests for changing the life cycle state of a Self-Description. In the case of a transition to "deprecated", a newer Self-Description for the same subject MUST be provided. Otherwise, a hashcode MUST be provided.	Each transition defined in the state diagram in Section 2.3 MUST be implemented according to the "workflow of changing the life cycle state of SDs" given in Figure 7.	Mandatory (Phase 1)

Table 12: *Self-Description Storage Subsystem Functions - Changing the Life Cycle State of Self-Descriptions*

4.1.2.4 Retrieving Self-Descriptions

ID	Description	Acceptance Criteria	Priority
SD-S-R-01	The Federated Catalogue MUST answer requests for Self-Descriptions with OPTIONAL filter criteria in terms of Administrative Metadata, e.g., the date when it was received by the Catalogue. "active" MUST be used as the default value for the life cycle *state*. The filtering expression language MUST allow for the union of multiple *state*s, as well as a range of dates. A number to limit the number of results MAY be provided.	A request without filter criteria MUST result in a list of Administrative Metadata of Self-Descriptions. In the absence of a limit, the result MUST include all Self-Descriptions in the Catalogue. If filter criteria were given, the result MUST include all matching Self-Descriptions. If a limit was given, the length of the result list MUST be limited to that number. It is not specified what potential results to include in the limited list.	Mandatory (Phase 1)
SD-S-R-02	The Federated Catalogue MUST answer requests for the Administrative Metadata of active Self-Descriptions that match a query that results in a set of Participants or Assets. This request has the same result type as SD-S-R-01 but takes as input a query rather than filter criteria. The query SHALL be executed according to Q-F-01.	The set of Self-Description subject IDs in the Administrative Metadata returned by this query MUST equal the set of Self-Description IDs in the result set of the query according to Q-F-01. To be tested at least with a query that is expected to return the empty set and a query that is expected to return a non-empty set.	Mandatory (Phase 1)
SD-S-R-03	The Federated Catalogue MUST answer requests for Self-Descriptions as in SD-S-R-01 or SD-S-R-02, but returning the results as raw JSON-LD documents with Administrative Metadata	A request for Self-Descriptions with Administrative Metadata must follow the Result format for Raw JSON-LD Self-Descriptions with Administrative Metadata as specified above in Listing 4.1.	Mandatory (Phase 1)

Table 13: *Self-Description Storage Subsystem Functions - Retrieving Self-Descriptions*

4.1.3 Provided Internal Interfaces

Read/Write access to the raw Self-Descriptions and lifecycle metadata.

Basic Query capabilities on the Self-Description lifecycle metadata.

4.1.4 Consumed Internal Interfaces

The Self-Description Storage Subsystem consumes the following internal interfaces:

- Interface of the Semantics and Trust Verification Subsystem, to validate the cryptographic signatures of Self-Descriptions (ST-V-T-01), and to perform semantic validation (ST-V-V-03)
- Interface to the query execution specified in Q-F-01 in Section 4.4

4.1.5 Functional Requirements

ID	Description	Acceptance Criteria	Priority
SD-S-F-01	The Federated Catalogue MUST perform syntactic validation (SD-S-V-01) on any Self-Description that is being submitted or updated and reject non-well-formed input with HTTP error code 400 Bad Request.	Unit test with a well-formed JSON-LD and a non-well-formed Self-Description implemented	Mandatory (Phase 1)
SD-S-F-02	The Federated Catalogue MUST perform semantic validation (ST-V-V-03) on any Self-Description that is being submitted or updated and that is syntactically well-formed (SD-S-F-01). Invalid input must be rejected with HTTP error code 422 Unprocessable Entity.	Unit test with a well-formed Self-Description that validates against the Self-Description Schema and another well-formed one that does not validate	Mandatory (Phase 1)
SD-S-F-03	The Federated Catalogue MUST continually check the temporal validity of the digital certifications contained in the Self-Description.	Any Self-Description with an expired certificate MUST be transitioned to the “revoked” state.	Mandatory (Phase 1)
SD-S-F-04	The Federated Catalogue MUST keep track of all Schema terms that are used by Self-Descriptions, i.e., properties (used as predicates of RDF triples, regardless of whether declared in a Schema known to the Catalogue), classes (used as objects in RDF triples whose predicate is *rdf:type*, regardless of whether declared in a Schema known to the Catalogue), and individuals (only those declared in a Schema known to the Catalogue). This functionality MAY be implemented using the same graph database that is also used by the Graph and Query Subsystem.	Unit test that adds a Self-Description and subsequently queries the database that keeps track of Schema term usage to ensure that the terms used in the Self-Description have been added.	Mandatory (Phase 1)

***Table 14**: Self-Description Storage Subsystem Functional Requirements*

4.1.6 Non-Functional Requirements

ID	Description	Acceptance Criteria	Priority
SD-S-NF-01	Requests for changing the life cycle state of a Self-Description by any Participant that is not the subject of the proof (signature) of the Verifiable Presentation that constitutes the	Unit test with requests from the Participant who originally added the Self-Description to the Catalogue (to be accepted) and a	Mandatory (Phase 1)

	given Self-Descriptions MUST be rejected with HTTP error code 401 Unauthorized.	different Participant (to be rejected)	
SD-S-NF-02	After adding or updating a Self-Description of a subject, the Federated Catalogue MUST return, when asked for the Self-Description of that same subject, the same Self-Description that was submitted with the add (SD-S-S-01) or update (SD-S-U-01) operation.	Add or update a Self-Description, then execute one of the SD-S-R-* retrieval operations, which is expected to return exactly that Self-Description. The returned raw Self-Description MUST literally match the JSON-LD document that was submitted.	Mandatory (Phase 1)
SD-S-NF-03	The Federated Catalogue MUST be able to store multiple versions of the Self-Description of the same subject.	Execute SD-S-U-01 to perform the transition of a Self-Description S of some subject to "deprecated" and update it by a new Self-Description S'. When using SD-S-R-01 to retrieve all Self-Descriptions of that subject, the request MUST return at least S in state "deprecated" and S' in state "active".	Mandatory (Phase 1)
SD-S-NF-07	About any subject, there MUST be at most one Self-Description with life cycle state "active" in a Federated Catalogue.	Invoke SD-S-S-01 or SD-S-U-01 with a Self-Description that has been added before, then execute SD-S-R-01 to retrieve all active Self-Descriptions about the same subject. Make sure that exactly one Self-Description is returned.	Mandatory (Phase 1)

Table 15: *Self-Description Storage Subsystem Non-Functional Requirements*

4.2 Schema Storage Subsystem

4.2.1 Description

For validating Self-Descriptions, the Federated Catalogue needs to know the schema to validate against. The primary schema is the Gaia-X Self-Description Schema. Supporting the Self-Description Core Ontology (cf. Appendix A) is mandatory for the Federated Catalogue. Further standard Gaia-X Self-Description Schemas Specific ecosystems, specific data spaces, etc., may require additional attributes beyond those defined in the Gaia-X Self-Description Schema. Thus, the Federated Catalogue shall be able to store additional Self-Description Schemas.

Such additional schemas must not define terms in the Gaia-X namespace http://w3id.org/gaia-x/core#. This namespace is reserved for the Gaia-X AISBL. Serializations, e.g., in JSON-LD, must not bind the *gax* prefix to any namespace other than http://w3id.org/gaia-x/core#.

Technically, Self-Description Schemas have the same format as Self-Descriptions. I.e., they are RDF graphs serialized as JSON-LD. A Self-Description Schema is made up of the following components:

1. An **ontology for defining the terms** (mandatory): the terms of a schema, i.e., entity types (classes) and the attributes (properties) that such entities may hold, as well as certain instances (individuals) of such entity types, are formally defined and documented as classes and properties in ontologies.

The Federated Catalogue shall support ontologies expressed in RDF Schema and/or the OWL Web Ontology Language.

2. **Shapes to validate against** (mandatory): for the purpose of validation, the expected usage of classes and properties in Self-Descriptions shall be defined in shapes in the SHACL Shapes Constraint Language.
3. **Controlled vocabularies** (optional[9]): to provide Self-Description authors guidance regarding frequently used attribute values, and to avoid ambiguous or underspecified attribute values, it may be helpful to define controlled vocabularies, from which the values of certain attributes should be drawn, rather than being expressed as free-form strings. Every term in a controlled vocabulary has a unique identifier and a semantically structured definition, like a term in an ontology. The Federated Catalogue shall support controlled vocabularies modelled as concept schemes in the SKOS Simple Knowledge Organization Scheme.

Syntactically, RDF Schema and OWL ontologies, SHACL shapes and SKOS controlled vocabularies are, once more, RDF graphs, for which the Federated Catalogue should support the JSON-LD serialization.

The Federated Catalogue stores Schemas in the same way as explained for Self-Descriptions in Section 4.1 (minus the validation of cryptographic signatures, which are not required for Schemas), applies Graph and Query functionality to them as specified in Section 4.4, as well as Inter-Catalogue Synchronization as specified in Section 4.5. In Phase 1, two features, which are mandatory for Self-Description Storage, are not yet mandatory for Schema Storage. This is to reduce management overhead.

- The Federated Catalogue is not required to provide a public API for submitting and updating Schemas. Instead, a Catalogue Administrator user with administrative permissions may upload the respective files.
- The Federated Catalogue is not required to retain previous versions of a Schema beyond the execution of function SD-Sch-F-05 as specified below.

In Phase 1, it is therefore possible that a Catalogue, after some Schema updates, contains Self-Descriptions that are not valid w.r.t. the Schema(s) stored in the Catalogue (But see SD-Sch-NF-05 for Phase 2.).

[9] The Federated Catalogue is required to support controlled vocabularies. However, not every Self-Description Schema is required to comprise a controlled vocabulary.

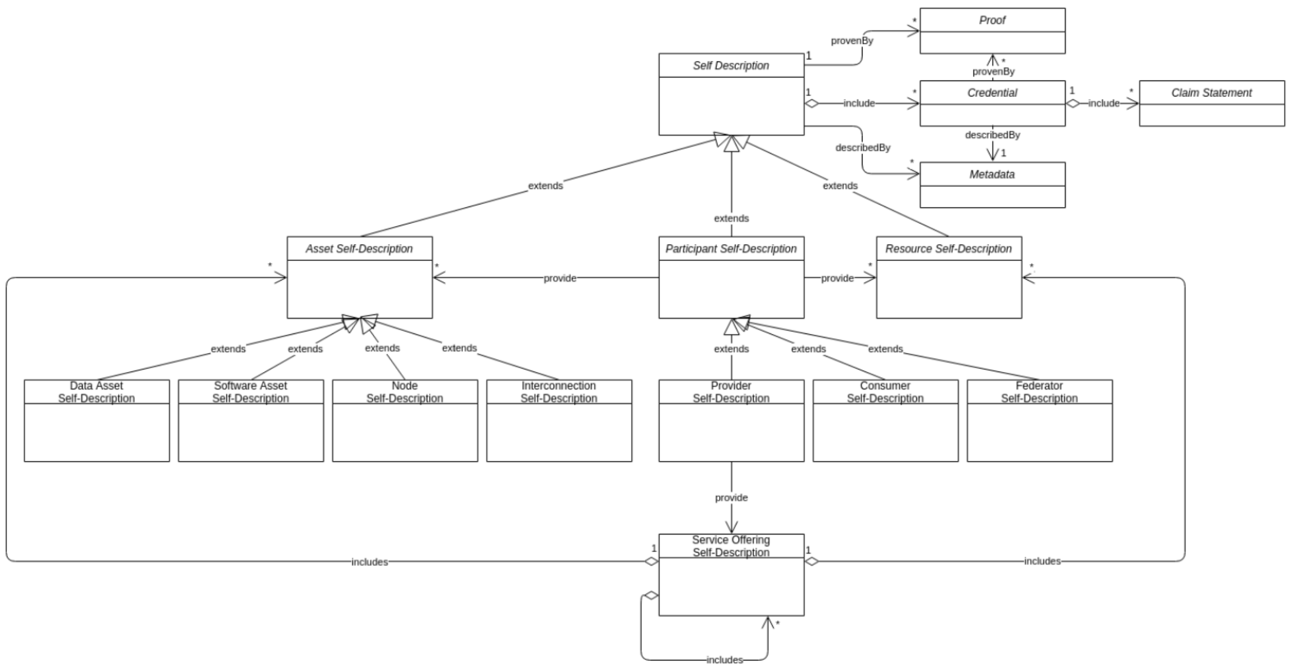

Proof
provenBy
Self Description
provenBy
Credential
include
Claim Statement
describedBy
describedBy
Metadata
extends
extends
extends
Asset Self-Description
Participant Self-Description
Resource Self-Description
provide
provide
Data Asset Self-Description
Software Asset Self-Description
Node Self-Description
Interconnection Self-Description
Provider Self-Description
Consumer Self-Description
Federator Self-Description
provide
Service Offering Self-Description
includes
includes
includes

***Figure 8**: Schematic inheritance relations and properties for the top-level Self-Description*

4.2.2 Functions

ID	Description	Acceptance Criteria	Priority
SD-Sch-01	Catalogue Administrators MUST be able to submit Self-Description Schemas.	Carry out manual submission of a Schema	Mandatory (Phase 1)
SD-Sch-02	Participants MUST be able to submit Self-Description Schemas like with Self-Descriptions using SD-S-S-01 (minus the validation of cryptographic signatures).	Cf. SD-S-S-01 (minus the validation of cryptographic signatures).	Optional (Phase 2)
SD-Sch-03	Catalogue Administrators MUST be able to delete Self-Description Schemas. (Phase 1 only)	Carry out manual deletion of a Schema	Mandatory (Phase 1)
SD-Sch-04	The Federated Catalogue MUST accept requests for changing the life cycle state of a Self-Description Schema from the same Participant who submitted the Schema, using SD-S-U-01 (minus the validation of cryptographic signatures). (In the case of updating/deprecating Schemas, "same-ness" is determined by the Ontology IRI.)	Cf. SD-S-U-01 (minus the validation of cryptographic signatures).	Optional (Phase 2)
SD-Sch-05	The Federated Catalogue MUST be able to return the Union Graph of all Self-Description Schemas, defined as the union of the RDF graphs of the Self-Description Core Ontology as well as all additional Self-Description Schemas (including Controlled Vocabularies) that have been added to the Catalogue. This functionality MAY be implemented using the same graph database that is also used by the Graph and Query Subsystem. (Note that all the other	Test with a triple that occurs in an individual Schema: This triple MUST also occur in the Union Graph.	Mandatory (Phase 1)

	retrieval functions SD-S-R-* apply to Schemas as well.)		

Table 16: Schema Storage Subsystem Functions

4.2.3 Provided Internal Interfaces

SD-Sch-05 is exposed to the Semantics and Trust Verification Subsystem.

4.2.4 Consumed Internal Interfaces

The Schema Storage Subsystem consumes the following internal interfaces in a Catalogue to implement necessary actions:

- Of SD-S-S-01 and SD-S-U-01 in the Self-Description Storage Subsystem, the sub-steps that do not involve cryptographic signatures, are exposed to the Schema Storage Subsystem.
- SD-V-V-03 from the Semantics and Trust Verification Subsystem is used for validation.

4.2.5 Functional Requirements

ID	Description	Acceptance Criteria	Priority
SD-Sch-F-01	The Federated Catalogue MUST be pre-loaded with the Self-Description Core Ontology as given in Appendix A.	Queries for key terms of the Self-Description Core Ontology	Mandatory (Phase 1)
SD-Sch-F-02	The Federated Catalogue MUST accept the submission of Schemas that are valid instances[10] of the RDF Schema, OWL, SHACL or SKOS specifications and that are represented as RDF graphs in the JSON-LD serialization.	Acceptance of the JSON-LD version of the Self-Description Core Ontology	Mandatory (Phase 1)
SD-Sch-F-03	The Federated Catalogue SHOULD accept the submission of Schemas that are valid instances of the RDF Schema, OWL, SHACL or SKOS specifications and that are represented as RDF graphs in any serialization other than JSON-LD, e.g., RDF/XML or Turtle.	Acceptance of the Self-Description Core Ontology, converted to RDF/XML or Turtle	Mandatory (Phase 1)
SD-Sch-F-04	The submission of a Schema S’ that has the same Ontology IRI as a Schema S that is already present in the Federated Catalogue SHALL result in S being deprecated by S’. (fulfilled by SD-Sch-04 in Phase 2)	Submit Schema S, then another Schema S’ with the same Ontology IRI.	Mandatory (Phase 1)
SD-Sch-F-05	Once having accepted the submission of a Schema S’ or executing the deletion of a Schema S’, the Federated Catalogue MUST validate (by applying SD-V-V-03) each Self-Description in the Catalogue that uses terms from S’ according to SD-S-F-04. If S’ deprecates a previous Schema S according to SD-Sch-F-04, the Federated Catalogue MUST also validate any additional Self-Descriptions (if any) that use terms from S. Self-	Test with a Schema S’ that invalidates a Self-Description in the Catalogue	Mandatory (Phase 1)

[10] Acknowledging the existence of dedicated libraries for these languages, we do not mandate validation to be performed like the validation of a Self-Description against a Schema (cf. ST-V-V-03) but require validation according to the specifications of the respective languages.

	Descriptions that fail the validation MUST be flagged invalid, e.g., via a respective Administrative Metadata field for internal purposes.		
SD-Sch-F-06	The Federated Catalogue MUST keep track of what versions of what Schemas a valid Self-Description was validated against.	Unit test that adds Schema S, a Self-Description SD using terms from S, and subsequently queries the database that keeps track of validation to ensure that it contains the information that S is a valid instance of SD.	Optional (Phase 2)

Table 17*: Schema Storage Subsystem Functional Requirements*

4.2.6 Non-Functional Requirements

ID	Description	Acceptance Criteria	Priority
SD-Sch-NF-01	The Federated Catalogue MUST NOT accept the submission of a Schema that does not have an Ontology IRI as specified by OWL [9].	Test with a Schema that has no Ontology IRI	Mandatory (Phase 1)
SD-Sch-NF-02	The Federated Catalogue MUST NOT accept the submission of a Schema that does not have a Version IRI as specified by OWL [9].	Test with a Schema that has no Version IRI	Optional (Phase 2)
SD-Sch-NF-03	Other than during an update of a Schema with a new version (i.e., having the same Ontology IRI), the Federated Catalogue MUST NOT accept the submission of a Schema that redefines terms that have already been defined in this Catalogue. The submission of SHACL shapes, which apply (e.g., via *sh:targetClass* or *sh:path*) to classes or properties for which only RDF Schema or OWL axioms have so far been defined in the Federated Catalogue does not count as a “redefinition”. This is to allow for the common practice of storing SHACL shapes in files separate from the ontology.	Submit a Schema S that defines a certain term, then submit a Schema S’ with a different Ontology IRI that defines the same term.	Mandatory (Phase 1)
SD-Sch-NF-04	The Federated Catalogue MUST NOT accept from a Participant the submission of a Schema that defines terms in the reserved http://w3id.org/gaia-x/core# namespace or whose serialization binds the *gax:* prefix to any namespace other than that. This SHOULD be implemented as soon as SD-Sch-S-02 has been implemented.	Test with a Schema that defines terms in the http://w3id.org/gaia-x/core# namespace or whose serialization binds the *gax:* prefix to any namespace other than that.	Optional (Phase 2)
SD-Sch-NF-05	At any point in time, for each Self-Description in the Catalogue whose life cycle state is “active”, there MUST be a Schema in the Catalogue of which the Self-Description is a valid instance.	Test with a Self-Description and a sequence of Schema updates	Optional (Phase 2)

Table 18*: Schema Storage Subsystem Non-Functional Requirements*

4.3 Semantics and Trust Verification Subsystem

4.3.1 Description

The Semantics and Trust Verification Subsystem performs two complementary activities to ensure a high quality of Self-Descriptions:

1. it validates Self-Descriptions against a Schema, thus assuring a certain level of data quality, and
2. it establishes trust into Self-Descriptions by verifying the cryptographic signatures attached to it.

Validation against the kind of Schemas introduced in Section 4.2 is more than just checking for syntactic well-formedness. The expressive power of SHACL additionally supports lightweight semantics.

Regarding **Trust**, the Federated Catalogue needs to be able to check that any Self-Description added to the Catalogue has a cryptographic proof and that this proof is correct – in other words: that the claims made about the subject of a Self-Description have been confirmed by someone (e.g., by the Provider or by an external evaluation facility) and have not been tampered with afterwards. Note that the correctness of a proof does not mean that the corresponding claims are true. When issues arise during the verification of trust of a Self-Description added to the Catalogue, the indication of failure may be returned to the registering Participant.

Verification of the trust in a Self-Description ultimately resides at the end user who is given access to the raw Self-Descriptions in JSON-LD format. The Catalogue internally performs trust verifications in two cases:

3. When it receives a Self-Description through the REST API or the Inter-Catalogue Synchronization. This acts as a filter where untrusted Self-Descriptions are rejected.
4. When a user requests to do a verification on a Self-Description supplied by the user.

The major verification steps for trust are:

5. Verify that a proof (i.e., a signature) is present on the overall Verifiable Presentation, e.g., the outer structure of every Self-Description.
6. Validate that the JSON-LD document of the Verifiable Presentation was not altered by verifying its proof according to the respective proof suite verification algorithm, as specified by the Verifiable Credentials Data Model [5]. This typically involves retrieving the public key of the issuer of the Self-Description.
7. From within the Self-Description, determine the identifier of the provided of the thing that is described. Every possible class from the Self-Description schemas (except for Participants themselves) references the provider-Participant via the “providedBy” property.
8. Verify, if possible, if the manager has a Self-Description as a Participant using the same identifier.
9. Verify that the manager is also the issuer of the Self-Description and has signed the overall Self-Description.
10. Verify that the issuer’s certificate has not been revoked.
11. Verify that the certificates of further signers of the Self-Description have not been revoked.
12. Verify that the expiration date is in the future.

All these steps are repeated for each Verifiable Credential inside the Verifiable Presentation.

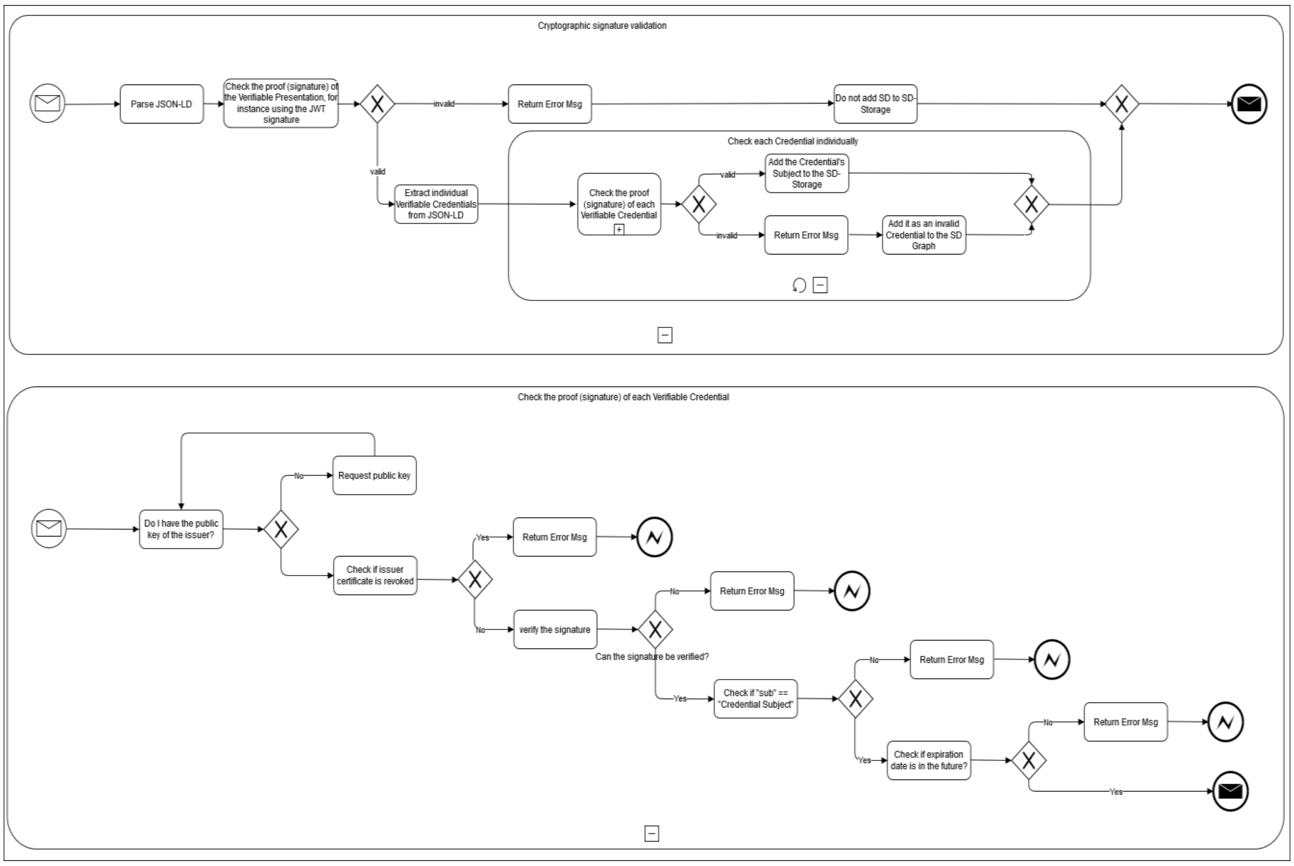

Cryptographic signature validation
Parse JSON-LD
Check the proof (signature) of the Verifiable Presentation, for instance using the JWT signature
invalid
Return Error Msg
Do not add SD to SD-Storage
valid
Extract individual Verifiable Credentials from JSON-LD
Check each Credential individually
Check the proof (signature) of each Verifiable Credential
valid
Add the Credential's Subject to the SD-Storage
invalid
Return Error Msg
Add it as an invalid Credential to the SD Graph
Check the proof (signature) of each Verifiable Credential
Do I have the public key of the issuer?
No
Request public key
Check if issuer certificate is revoked
Yes
Return Error Msg
No
verify the signature
No
Return Error Msg
Can the signature be verified?
Yes
Check if "sub" == "Credential Subject"
No
Return Error Msg
Yes
Check if expiration date is in the future?
No
Return Error Msg
Yes

Figure 9: *Workflow of details of Cryptographic signature validation*

4.3.2 Functions

4.3.2.1 Validating a Self-Description against the Schema

ID	Description	Acceptance Criteria	Priority
ST-V-V-01	The Federated Catalogue MUST be able to extract, from the representation of a Self-Description as a Verifiable Presentation, the plain, unsigned Self-Description, i.e., the union of all *credentialSubject*s. This relies on all Credentials having the same subject.	A query that checks whether the set of all triples nested into *verifiableCredential/credentialSubject* in the Verifiable Presentation equals the set of all triples extracted into the plain, unsigned Self-Description	Mandatory (Phase 1)
ST-V-V-02	The Federated Catalogue MUST be able to validate a plain, unsigned Self-Description against a SHACL shapes graph.	A raw Self-Description is a valid instance of a SHACL shapes graph if it validates against the SHACL shapes of the Self-Description Schema in the sense of "Validation of a data graph against a shapes graph" in Section 3.4 of the SHACL specification.	Mandatory (Phase 1)
ST-V-V-03	The Federated Catalogue MUST be able to validate a Self-Description, which is represented as a Verifiable Presentation, against the Union of all Self-Description Schemas (computed according to SD-Sch-05).	(proper composition of functions ST-V-V-01, SD-Sch-05, and ST-V-V-02)	Mandatory (Phase 1)

Table 19*: Validating a Self-Description against the Schema*

4.3.2.2 Trust Verification

ID	Description	Acceptance Criteria	Priority
ST-V-T-01	The Federated Catalogue MUST provide a function to validate the cryptographic signatures of a Self-Description given as a JSON-LD Verifiable Presentation.	"Workflow of details of Cryptographic signature validation" implemented according to Figure 9	Mandatory (Phase 1)

Table 20*: Semantics and Trust Verification Subsystem Trust Verification*

4.3.3 Provided Internal Interfaces

SD-V-V-01 is exposed to the Self-Description and Schema Storage modules.

4.3.4 Consumed Internal Interfaces

Schema validation uses function SD-Sch-05, which computes the Union of all Self-Description Schemas.

4.3.5 Functional Requirements

ID	Description	Acceptance Criteria	Priority
ST-F-01	The Federated Catalogue MUST be able to verify proofs.	As specified by the Verifiable Credentials Data Model [5].	Mandatory (Phase 1)
ST-F-02	The Federated Catalogue MUST be able to retrieve the public key of a Participant.	We do not prescribe a specific mechanism for publishing and retrieving public keys but point out the possibilities of using keyservers, which associate with a participant's identity its public key, or embedding a Participant's public key into its Self-Description using a suitable RDF property[11].	Mandatory (Phase 1)

Table 21: *Semantics and Trust Verification Subsystem Functional Requirements*

4.4 Self-Description Graph and Query Subsystem

4.4.1 Description

The different Self-Descriptions of the Federated Catalogue need to be stored into one consistent database to enable the connection of elements in the different Self-Descriptions. For instance, several Self-Descriptions can point to a unique element, e.g., a certain certification level, which may be a starting point for a discovery approach. A user of the Federated Catalogue might be interested to start at the certification level, and query for applying services. This is only possible when all information is indexed in one entity. This entity is called the Self-Description Graph and defined with the following requirements of Section 4.4.2.

[11] See, e.g., *cert:key* from the Certificates ontology, a W3C Editor's Draft. https://www.w3.org/ns/auth/cert#key

The SD Graph provides an efficient indexing of the loaded SDs. As such, this index is the backbone for the fast and efficient answering of the incoming queries. It is in the responsibility of the Catalogue operator how this index is constructed and maintained as long as the reliable answering of queries is ensured.

As for the query processing over the Self-Description Graph in the Catalogue, input openCypher queries are in compliant with Self-Description Schema and the query results are serialized in JSON by default but can be delivered also in further data formats, for instance XML, CSV, or HTML. Therefore, certain parameters must be predefined in HTTP header when sending the query requests. Through queries, data consumers can retrieve expected results and the query process is illustrated in Figure 10.

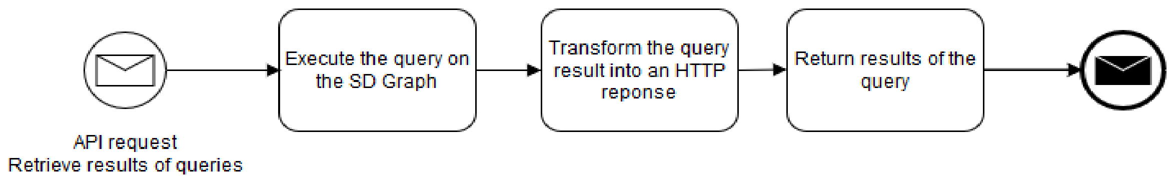

Execute the query on the SD Graph
Transform the query result into an HTTP reponse
Return results of the query
API request
Retrieve results of queries

***Figure 10**: Workflow of retrieving results of queries*

The Catalogue is able to receive openCypher queries through the remote API. The query endpoint is a well-known path and is defined in the normative OpenAPI file in Appendix B. The query endpoint can reject query requests that violate a set of transparent policies to protect the server stability. For instance, the query endpoint may abort the execution of a query after a certain time interval or reject any unsafe (create, delete) operation.

For valid queries, the query results must be returned in non-discriminating ranking, besides predefined filter criteria set by Data Consumer. Furthermore, the Federated Catalogue must have a log of all queries, and their content must be stored in an immutable way.

The Federated Catalogue may send queries also searching for SDs that contain one or more invalid or expired signatures, if the incoming request explicitly indicates that it also asks for not completely trustworthy metadata. As the Self-Description Storage stores the input of the Self-Description Graph persistently, which enables a client to send requests for any version or all versions of a Self-Description. The SD Graph itself provides two partitions. One that only contains the SDs, for which all signatures could be verified, and the other also provides SDs that (partly) contain all non-revoked SDs.

4.4.2 Functions

The Self-Description Graph and Query Module provides several interactions and operations that are defined in the following.

4.4.2.1 Functions of the Self-Description Graph

ID	Description	Acceptance Criteria	Priority
SD-G-F-01	The SD Graph must be able to add active SDs. The plain, unsigned content of the SDs is obtained using S-V-V-01.	The client queries the Catalogue for a new SD_1. The Catalogue returns no result. The client then sends the SD_1 to the Catalogue and in a second request queries for SD_1 again. Now, the Catalogue answers with the complete content of SD_1.	Mandatory (Phase 1)

SD-G-F-02	The SD Graph must be able to update active SDs. The plain, unsigned content of the SDs is obtained using S-V-V-01.	A previously registered SD_1 is updated by its successor SD_1'. Then the client now queries for the identifier of SD_1 without indicating that it is interested in the old version, the Catalogue responses with the content of SD_1'.	Mandatory (Phase 1)
SD-G-F-03	The SD Graph must be able to revoke SDs.	The client that previously added a certain SD executes a DELETE operation towards the identifier of the SD. A client requesting this specific SD later on receives a "not available" response.	Mandatory (Phase 1)
SD-G-F-04	The SD Graph must be readable for the user.	The query 'MATCH (<node:node_ID>) RETURN node.label' with an existing node ID returns a result.	Mandatory (Phase 1)

***Table 22**: Functions of the Self-Description Graph*

4.4.2.2 Functions of the Query Module

ID	Description	Acceptance Criteria	Priority
Q-F-01	The Query module must enable a user to find offerings that match their requirements by executing openCypher queries.	Execute an example openCypher query and matched results can be correctly returned.	Mandatory (Phase 1)
Q-F-02	The Query module must be able to return the cryptographic proof of a SD, i.e., of the Verifiable Presentation.	Verification steps must align with the content in 4.3.1	Mandatory (Phase 1)
Q-F-03	The Query module must be able to serialize the query results in JSON by default.	The request data format of the query results as an IANA mimetype. Default is 'application/json'.	Mandatory (Phase 1)
Q-F-04	The Query module must be able to predefine necessary parameters in HTTP header.	The parameters should be aligned with xxx.	Mandatory (Phase 1)

***Table 23**:Functions of the Query Module*

4.4.3 Provided Internal Interfaces

SD Graph and Query subsystem provides internal interfaces to the following modules in a Catalogue to implement necessary actions:

- Provide an internal interface to Self-Description Storage to accept "active" SDs into SD Graph
- Provide an internal interface to Catalogue REST API to execute queries over the SD Graph

4.4.4 Consumed Internal Interfaces

SD Graph and Query subsystem consumes the following internal interfaces in a Catalogue to implement necessary actions:

- Consume an internal interface of Catalogue REST API to return the query results

4.4.5 Functional Requirements

4.4.5.1 Self-Description Graph

ID	Description	Acceptance Criteria	Priority
G-FR-01	SD Graph must set up backup.	100% data can be recovered after a crash. The backup functionality is continuous.	Mandatory (Phase 1)
G-FR-02	SD Graph must accept the verified Self-Descriptions.	A verified Self-Description needs a valid certification and the syntactic and semantic correctness against a given schema.	Mandatory (Phase 1)
G-FR-03	SD Graph must allow updates of a registered Gaia-X Self-Description if the update request was originally initiated by the participant of this Self-Description, or by the Gaia-X entity controlling the participant, if this relation has been made visible to the Federated Catalogue.	A Self-Description S of a Gaia-X Service A with version n can be updated with a new Self-Description S' for A, where S is different to S'. After the update, the attributes of S'are shown to a requesting client.	Mandatory (Phase 1)
G-FR-04	SD Graph must not present data of a removed or passivated component after its removal has been acknowledged to the requesting entity.	SDs in other life cycle statues except "active" must not be presented.	Mandatory (Phase 1)
G-FR-05	A client can query the latest version of a Self-Description Graph.	The latest version number of Self-Description Graph should be presented in query.	Mandatory (Phase 1)
G-FR-06	The SD Graph needs to be recreated from the stored SD files in periodic intervals and whenever the used schema has been updated.	The SD Graph has a feature that reimports all SDs from the SD Storage and the manual run of this feature results in a correct SD Graph.	Mandatory (Phase 1)
G-FR-07	A Catalogue must allow the subscription of clients to changes in the SD module.	The Catalogue offers an API for clients to subscribe to specific SDs using the hash identifier of the targeted SD.	Optional (Phase 2)
G-FR-08	A Catalogue must announce (publish) creations and changes to SDs to subscribed clients.	Client subscribes to one specific SD, the SD is changed to SD' and the client receives an appropriate message, containing SD'.	Optional (Phase 2)

Table 24: *Self-Description Graph Functional Requirements*

4.4.5.2 Query Module

ID	Description	Acceptance Criteria	Priority
Q-FR-01	The result of an openCypher query must contain a reference to the originating SD for each returned result item.	The "meta" element of the "results" JSON object contains the unique hash of the SD that	Mandatory (Phase 1)

		contained the original information.	
Q-FR-02	A Catalogue must not allow unsafe operation through the query module.	Send openCypher queries with the CREATE, REMOVE, DELETE, and LOAD clauses. All must be rejected.	Mandatory (Phase 1)
Q-FR-03	Executing queries needs to be in compliance with Self-Description Schema.	SHACL validation against Self-Description Schema must pass.	Mandatory (Phase 1)
Q-FR-04	Responses are also in compliance with Self-Description Schema.	SHACL validation against Self-Description Schema must pass.	Mandatory (Phase 1)
Q-FR-05	The result of an openCypher query must be paginated.	The result of an openCypher query uses pagination for batches of 100 return items.	Mandatory (Phase 2)

Table 25*: Functional Requirements Query Module*

4.4.5.3 Query Parameters

ID	Name	Description	Priority
Q-Par-01	query-language* string (header)	Available values: openCypher, application/sparql-query, sparql*; Default value:openCypher	Mandatory (Phase 1)
Q-Par-02	Self-Description-schemaVersion string (header)	Self-Description Schema version, against which the Message should be interpreted.	Mandatory (Phase 1)
Q-Par-03	federatedCatalogue-issued ($date-time) (header)	Date of issuing the request.	Mandatory (Phase 1)
Q-Par-04	federatedCatalogue -senderAgent string($uri) (header)	*gax:Participan*t, which initiated the message.	Optional (Phase 1)

Q-Par-05	federatedCatalogue -securityToken * string (header)	The Gaia-X DAT: Token representing Gaia-X security claims, for instance that the sender supports a certain security profile.	Optional (Phase 2)
Q-Par-06	federatedCatalogue -authorizationToken string (header)	Authorization token as required by the Federated Catalogue.	Optional(Phase 2)
Q-Par-07	federatedCatalogue -transferContract string($uri) (header)	Contract which is (or will be) the legal basis of the data transfer.	Mandatory (Phase 2)
Q-Par-08	Accept	The request data format of the query results as an IANA MIME type. Default is 'application/json'.	Optional (Phase 1)

Table 26*: Functional Requirements Query Parameters*

4.4.6 Non-Functional Requirements

4.4.6.1 Self-Description Graph

ID	Description	Acceptance Criteria	Priority
G-NF-01	The indexed graph must be capable of handling at least one million defined nodes, ten million properties, and ten million annotations at nodes or properties.	Prepare a dataset with one million defined nodes, ten million properties, and ten million annotations at nodes or properties, load it into the graph index and execute an openCypher query that returns the information at one explicit node.	Mandatory (Phase 1)
G-NF-02	Self-Description Graph needs to be restarted in a reasonable time limit after shutdown.	Self-Description Graph needs to be restartable in less than 2 minutes after a shutdown.	Mandatory (Phase 1)
G-NF-03	Self-Description Graph must handle the creation, update, and deletion of a single SD in less than 2 seconds.	Execute the creation, update, and deletion of SD_i (0 <= i <= 9) at least 100 times and measure the maximum execution time for each iteration. The criteria is	Mandatory (Phase 1)

		fulfilled if MaxExecutionTime <= 2 sec.	

Table 27: Self-Description Graph Non-Functional Requirements

4.4.6.2 Query Module

ID	Description	Acceptance Criteria	Priority
Q-NF-01	The query functionality must be able to handle 100 single read events in one second. An atomic activity is an operation that targets only one entity of the graph.	Execute the template 'MATCH (<node:node_i>) RETURN node.label' for 0 <= i <= 9 at least 100 times and measure the maximum response time for each iteration. The criteria is fulfilled if MaxResponesTime <= 1 sec.	Mandatory (Phase 2)
Q-NF-02	The query functionality must be able to handle an average of 20 query requests per second.	Execute the template 'MATCH (<node:node_i>) RETURN node.label' for 0 <= i <= 19 at least 100 times and measure the average response time for each iteration directly at the Catalogue API, after the query message has reached the Catalogue and before it is sent to the client. The criteria is fulfilled if AverageProcessingTime <= 1 sec.	Mandatory (Phase 1)
Q-NF-03	A Catalogue must abort the query execution after 5 seconds and announce the timeout in the response message to the client.	Execute a query that takes longer than 5 seconds and check whether the execution is aborted after 5 seconds.	Mandatory (Phase 1)
Q-NF-04	There must be a limit of accuracy of returned results.	The accuracy of returned results is higher than 99%.	Mandatory (Phase 1)

Table 28: Non-Functional Requirements Query Module

4.5 Catalogue REST API Subsystem

4.5.1 Description

A Catalogue instance must provide a public REST API to manage the Objects within the Catalogue. There are two ways to use the REST API:

a) using it as an anonymous visitor or
b) using it as a registered user.

As a visitor only reading actions are allowed. The permissions of a registered user depend on his assigned roles. For the access rights of the provided roles have a look in the Role Activity Matrix in Section 2.2.

The REST API must be documented in a well-structured format e.g., OpenAPI 3.0 standard.

The entire REST API must support TLS encryption with at least version 1.3.

A Catalogue responds in defined manners. The API uses standard HTTP status codes and, whenever necessary, explains the result of the operation in English terms. Support for additional languages is optional.

The Federated Catalogue may require incoming clients to prove their identity. A Federated Catalogue may reject access to anonymous or wrongly identified clients. The rejection will be announced through an appropriate HTTP response code and error message.

Authentication and authorization of a request are enabled by Federated Catalog API. Requests to the Federated Catalogue API must be authorized using the username and access token of a valid user in the Catalogue. An authorization must be presented in a request header, otherwise, the server of the Catalogue must reply with an error indicating that no authentication header provided. When an incorrect authentication is supplied, an error message expressing invalid username and password must be replied.

An incomplete example REST API could be found in the Appendix B. It is based on OpenAPI 3.0 and could be well formatted with some tools for instance with swagger.io.

4.5.2 Functions

The following subchapters describe the activities which can be triggered by the REST API. The activities are listed in the tables below. An activity performs an action on an Object. Following Objects are captured in the Catalogue:

- Participants
- Principals / Users
- Self-Description
- Query
- Schema Management
- Roles

Each API Call has an operation identifier which is used in the REST API specification. If an activity leads to an API call, then the affiliated operation id is mentioned in the description column.

Each activity only can be performed by a user with the appropriate roles. Otherwise, the user gets an error message with an appropriate HTTP error code and the API call has no further impact on the Catalogue.

4.5.2.1 Participants

Participants of Gaia-X must be able to register themselves in a Federated Catalogue via the API. They may use platforms like the GXFS portal (a separate lot in the GXFS tender) . However, the REST API must provide participant management operations.

A participant is also described by a Self-Description. The structure of participants, the relationships to users and roles are described in Section 2.4.

ID	Description	Acceptance Criteria	Priority
Act-Pa-00	Add Participant - initial registration of an existing Participant to the Catalogue instance. API Operation ID: *addParticipant*	When a participant is successfully registered then the SD of the participant could be found in the Catalogue.	Mandatory (Phase 1)

Act-Pa-01	Read Participant - Get Self-Description of the requested participant. API Operation ID: *getParticipant, listParticipants*	If a participant is registered in the Catalogue, then its Self-Description will be replied.	Mandatory (Phase 1)
Act-Pa-02	Update Participant - Update the Self-Description of the requested participant. API Operation ID: *updateParticipant*	After a successful update of the Self-Description of a participant, the old Self-Description is replaced by the updated one. Only for standalone deployment of a Catalogue without an existing IAM system.	Mandatory (Phase 1)
Act-Pa-03	Delete Participant - Delete a participant from Catalogue. Deletion will not be done immediately e.g., latest after 30 days due to synchronization of SD storage and GraphDB. API Operation ID: *deleteParticipant*	The Self-Descriptions and users of the deleted participant are no longer stored in the Catalogue after the synchronization time.	Mandatory (Phase 1)
Act-Pa-04	Get all users of the participant. API Operation ID: *getUsersOfParticipant*	If a participant has 1 to n users, the Self-Descriptions of its 1 to n users will be replied.	Mandatory (Phase 1)

***Table 29**: Catalogue REST API Subsystem Functions Participants*

4.5.2.2 Principals / Users

For standalone deployment of a Catalogue without an existing IAM system the Federated Catalogue API must provide user management operations. Users with the appropriate roles need the possibility to create, read, update, and delete users in the Federated Catalogue.

A user is also described by a Self-Description and signed by the associated participant. The structure of users, the relationships to participants and roles are described in Section 2.4.

ID	Description	Acceptance Criteria	Priority
Act-Us-00	Create User - Add a user which belongs to a participant. The registration needs a valid Verifiable Credential signed by the participant. API Operation ID: *addUser*	After a successful adding of a user to a participant, the user can log in to the Catalogue with his credentials. The permissions of the user depend on his assigned roles.	Mandatory (Phase 1)
Act-Us-01	Read User - Get properties of requested user. API Operation ID: *getUser, listUsers*	If a user is registered in the Catalogue, then its Self-Description will be replied.	Mandatory (Phase 1)
Act-Us-02	Update User - Update properties of requested user. API Operation ID: *updateUser*	After a successful update of the Self-Description of a user, the old Self-Description is replaced by the updated one.	Mandatory (Phase 1)
Act-Us-03	Delete User - Delete a user from the participant in the Catalogue. API Operation ID: *deleteUser*	The user is deleted from the Catalogue. He can no longer log in to the Catalogue. He has no longer permissions to perform any activity in the Catalogue.	Mandatory (Phase 1)

***Table 30**: Catalogue REST API Subsystem Functions Principals/ Users*

4.5.2.3 Self-Description

The Self-Description documents contain the metadata about a service offering. Their serializations must be in JSON-LD and validate against the Gaia-X Self-Description Schema. These self-description serializations can be registered at a Federated Catalog. In addition, the related Provider must be able to update already registered Self-Descriptions whenever a change in the underlying offer appeared.

Sending a Self-Description using the Federated Catalogue API

A Provider can send Self-Descriptions about its offers to a Federated Catalogue, either via the remote API or using the website. The Federated Catalogue stores and indexes the Self-Description and makes it available to other Gaia-X Participants to query for them.

Synchronizing a new Self-Description from a remote Federated Catalogue

Whenever an interconnected Federated Catalogue receives a new or updated Self-Description, it will synchronize its new state with the other Federated Catalogue instances. This happens automatically and independently of the content of the Self-Description, its represented offering, or the registering Participant.

A Catalogue might as well reject Self-Descriptions with problematic signatures. In that case, it must announce at least one of the found errors to the registering Participant.

Sending delete request using the Federated Catalogue API

A registered Self-Description can be deleted by an authorized Participant, usually the one who created it or a Participant that is explicitly authorized to do so. In addition, dedicated Catalogue administrators have the permission to delete corrupted or outdated Self-Descriptions. Furthermore, they are allowed to delete any Self-Description that puts the proper functionality of the Catalogue at risk, for instance by their size.

A Deletion event results in the Self-Description in question not being listed anymore in the regular search queries. The Catalogue may however still store a local copy for legal issues or to provide backup capabilities.

Response Definition

A delete request must be responded in a corresponding HTTP status code via Federated Catalogue API. Whenever necessary, an explanation message of the refused request may also be replied in English terms. Support for additional languages is optional.

Sending access request using the Federated Catalogue API

Self-Descriptions can be accessed jointly. Through references between Self-Descriptions a federated query can retrieve all related Self-Descriptions with all signatures in this Catalogue.

The Catalogue must reply to an access request to many Self-Descriptions in a corresponding HTTP status code via Federated Catalogue API. In case an access request being rejected, or an access request to a certain Self-Description being rejected, a message may return to explain the rejection reason in English terms. Support for additional languages is optional.

Checking the Trust-Level/Signature of the Self-Description/Authorization

Before a Deletion request is accepted, the Catalogue must verify the authorization of the deleting Participant. If the Catalogue comes to the conclusion that the Participant is not authorized to delete the Self-Description, it must respond with a proper error message.

Each Self-Description can be accessed individually. If a Catalogue has the Self-Description in its storage and it has not been deleted before, it must return it together with all signatures. A client may however also indicate that it is also interested in Self-Description with a state different to Active. In that case, the Catalogue may or may not return the Self-Description.

In case a Self-Description has not been registered at a Catalogue, the Catalogue acts if it is not existing. A client therefore must not derive the non-existence of a Service Offering from an empty answer from one Catalogue. It is not in the responsibility of the Federated Catalogue to be aware of every existing Self-Description.

Synchronized Catalogues must announce the deletion of a Self-Description to each other. A Catalogue informed in such a way must behave as if the Participant had deleted the Self-Description directly at itself.

The Catalogue must validate Self-Descriptions syntactically and, to the extent covered by the Self-Description Schema, semantically. A non-compliant Self-Description must be rejected, and the Catalogue must reject it. Furthermore, the Catalogue must announce the rejection to the registering Participant, either via a proper response code (API) or error message on the website. A rejected Self-Description must not be added to the storage modules.

Gaia-X Self-Descriptions contain signatures of the creating entities. A Catalogue evaluates all signatures and indicates found issues to the registering Participant. Still, a Catalogue might decide to store a Self-Description where one or several signatures could not be verified. Such Self-Descriptions must be labeled as such as its trust level is reduced.

ID	Description	Acceptance Criteria	Priority
Act-SD-00	Add Self-Description - Add a Self-Description to the Catalogue. API Operation ID: *addSelfDescription*	When a SD is successfully added then the SD could be found in the Catalogue.	Mandatory (Phase 1)
Act-SD-01	Get Self-Description - getting the raw JSON-LD. API Operation ID: *readSelfDescriptionByHash, readSelfDescriptions*	If a Self-Description is publicly available in the Catalogue, then the Self-Description will be replied. Anybody could get every public SD he wants. The Catalogue can limit the rate to check for abuse.	Mandatory (Phase 1)
Act-SD-02	Update Self-Description - Update parts of a Self-Description. API Operation ID: *updateSelfDescription*	After a successful update of the Self-Description, the old Self-Description is marked as deprecated.	Mandatory (Phase 1)
Act-SD-03	Revoke Self-Description - Set the life cycle state of a Self-Description to revoked. API Operation ID: *updateSelfDescription*	The revoked Self-Description has the life cycle state "Revoked".	Mandatory (Phase 1)
Act-SD-04	Verify trust of a SD - Trust mechanism. Check the signatures of a SD with the information available in the Catalogue. API Operation ID: *verifyTrust*	If a Self-Description has invalid signatures, then this action will return a fail status code. If a Self-Description has valid signatures, then this action will return a success status code.	Mandatory (Phase 1)
Act-SD-05	Verify syntactic correctness of a SD - Check a SD to syntactic correctness	If a Self-Description does not comply with the current schema version of Self-Description, the action will fail.	Mandatory (Phase 1)

	API Operation ID: *verifySyntax*	If a Self-Description complies with the current schema version of Self-Description, the action will succeed. The verification is publicly available and can be used for debugging Self-Description. Therefore, responses with meaningful content will be replied.	

Table 31: *Catalogue REST API Subsystem Functions Checking the Trust-Level/Signature of the Self-Description/Authorization*

4.5.2.4 Query

A Gaia-X Catalogue must support the search for suitable Gaia-X Services using a HTTP API. The queries are stated in openCypher and serialized in the request with additional possible filter and search parameters.

The Federated Catalogue API allows the users to execute a series of openCypher queries to access Self-Descriptions. Against an HTTP API endpoint, the HTTP request with the openCypher query in the request body will be executed over the Self-Description Graph in the Catalogue. In the scenario of sending a query request, only the HTTP POST method is supported.

A successful response of a query request must include the corresponding HTTP response code and a reference to the originating SD for each returned result item.

For details to the queries see Section 4.4.

ID	Description	Acceptance Criteria	Priority
Act-Qu-04	Execute a Query - Sending Query to the Catalogue which should be executed to the Graph Database. API Operation ID: *query*	A valid query will get a valid and appropriate result.	Mandatory (Phase 1)

Table 32: *Catalogue REST API Subsystem Functions Query*

4.5.2.5 Schema Management

A Gaia-X Catalogue must provide the latest Self-Description schemas. This enables creating syntactic correct Self-Descriptions.

The structure of a schema is described in Section 4.2.

There are different resources / types which can be described by Self-Descriptions, e.g., Participants or DataAssets. Each type has its own schema, which must be available through the Federated Catalogue API. These schemas are stored in a graph-based structure and enable reuse of partial schemas whenever possible. Responding to a Self-Description schema request includes gathering all properties relevant for the requested Self-Description type from that graph and aggregating these into a (possibly hierarchical) resulting schema. For a given class *C*, this includes but is not limited to,

- from the ontology,
    - the immediate super classes,
    - the properties *P*, and
- from the shapes,
    - those that have *C* as their target class, and
    - those that have one property out of P in their path.

ID	Description	Acceptance Criteria	Priority
Act-Sc-01	Get the latest schemas of all types. API Operation ID: *getLatestSchemas*	For an API call, the Catalogue returns for each Self-Description type a valid assembled Self-Description schema that includes the latest state of all relevant properties belonging to that schema.	Mandatory (Phase 1)
Act-Sc-02	Get the latest schema of a specific type API Operation ID: *getLatestSchemasOfType*	Given an arbitrary type via an API call, the Catalogue returns a valid assembled Self-Description schema that includes the latest state of all relevant properties belonging to that schema.	Mandatory (Phase 1)

Table 33*: Catalogue REST API Subsystem Functions Schema Management*

4.5.2.6 Roles

For standalone deployment of a Catalogue without an existing IAM system, the Federated Catalogue API must provide role management operations. Users with the appropriate roles need the possibility to assign and revoke roles to other users.

The structure of roles, the provided set of roles and their definitions are described in Section 2.4. The access rights of the roles can be found in the table in Section 2.2.

ID	Description	Acceptance Criteria	Priority
Act-Ro-00	Assign / Revoke external roles to a user API Operation ID: *updateUserRoles*	After assigning an external role A to a user, the user has the permissions of role A. After revoking an external role B from a user, the user has no longer permission of role B.	Mandatory (Phase 1)
Act-Ro-01	Assign / Revoke internal roles to a user API Operation ID: *updateUserRoles*	After assigning an internal role A to a user, the user has the permissions of role A. After revoking an internal role B from a user, the user has no longer permission of role B.	Mandatory (Phase 1)
Act-Ro-02	Get roles of a user API Operation ID: *getUserRoles*	If roles A and B are assigned to a user, then a list containing the ids of role A and B is replied.	Mandatory (Phase 1)
Act-Ro-03	Get list of all possible roles API Operation ID: *getAllRoles*	All provided roles of the Catalogue will be replied. At least the default roles defined in section 2.4.	Mandatory (Phase 1)

Table 34*: Catalogue REST API Subsystem Functions Roles*

4.5.3 Provided Internal Interfaces

The Federated Catalogue API does not provide any internal interfaces.

4.5.4 Consumed Internal Interfaces

The Federated Catalogue API provides a public interface to perform actions on Objects in the Federated Catalogue by users. To delegate the request to the right place, the API consumes the provided internal interfaces of each Object which is captured by the API.

In particular, following internal interfaces are consumed by the REST API:

- User Management see Section 4.6 (for Objects: Participants, Principals / Users, Roles)
- SD Storage see Section 4.1 (for Object: Self-Description)
- SD Graph see Section 4.4 (for Object: Query)
- Schema Management see Section 4.2 (for Object: Schema)

4.5.5 Functional Requirements

ID	Description	Acceptance Criteria	Priority
Rest-F-01	The Catalogue MUST accept syntactically correct openCypher queries in the body of a HTTP POST request on the query endpoint.	Send a MATCH clause and verify whether it is returning content.	Mandatory (Phase 1)
Rest-F-02	The query endpoint MUST return a JSON object in the response body by default.		Mandatory (Phase 1)
Rest-F-03	The query endpoint SHOULD return a different data format, for instance XML, CSV, or HTML, if explicitly asked for by the client through Content Negotiation as defined by RFC 7231.	Execute query requests with the Accept header set to ‘application/xml’, ‘text/csv’, and ‘text/html’.	Optional (Phase 2)
Rest-F-04	The query endpoint SHOULD reject non-authorized clients or clients with incorrect identity claims.	Execute query requests with an arbitrary identity claim.	Optional (Phase 2)
Rest-F-05	The results of a query MUST NOT discriminate between different SDs or Providers. An ordering must be based on technical (for instance alphabetical or chronological sorting) or trust-related aspects (verified SDs first)	An equivalent SD is presented in the same way independent of the Participant that registered it.	Mandatory (Phase 1)
Rest-F-06	The results of a query MUST be delivered in batches of 100 result items by default. A client can use limit and offset parameters to request the distinct batches.		Mandatory (Phase 1)
Rest-F-07	The API MUST use standard HTTP status codes and, whenever necessary, explains the result of the operation in English terms. Support for additional languages is optional.		Mandatory (Phase 1)
Rest-F-08	There MUST NOT be different users with the same username under the same participant.	Creating a user with an already existing username under the same participant will be rejected.	Mandatory (Phase 1)
Rest-F-09	A Participant MUST NOT be registered more than once.	If a Participant wants to register which is already registered in Catalogue, it will be rejected.	Mandatory (Phase 1)
Rest-F-10	The signatures of a Participant MUST be valid during registration.	A Participant with an invalid or outdated signature cannot register.	Mandatory (Phase 1)

Rest-F-11	The Catalogue MUST NOT consider invalid or manually added role ids in the access token in the request header.	Invalid role ids in the access token are ignored.	Mandatory (Phase 1)
Rest-F-12	A user MUST NOT be assigned to one role more than once.	If a user is already assigned to a Role A and a new request wants to assign Role A again to the user, the request is rejected and has no further impact.	Mandatory (Phase 1)
Rest-F-13	Self-Descriptions which are uploaded to the Catalogue MUST match the latest schema.	Operations to upload or to adapt Self-Descriptions which leads to or contains invalid or outdated schemas are rejected.	Mandatory (Phase 1)
Rest-F-14	Self-Descriptions which are uploaded to the Catalogue MUST have valid and trustworthy signatures.	Operations to upload or to adapt Self-Descriptions with invalid and untrustworthy signatures are rejected.	Mandatory (Phase 1)
Rest-F-15	Query requests MUST have a valid structure.	Invalid queries will be rejected.	Mandatory (Phase 1)
Rest-F-16	Query requests with malicious content MUST be rejected.	A query which tries unauthorized to manipulate the Self-Description Graph, will be rejected.	Mandatory (Phase 1)

***Table 35**: Catalogue REST API Subsystem Functional Requirements*

4.5.6 Non-Functional Requirements

ID	Description	Acceptance Criteria	Priority
Rest-NF-01	The Catalogue MUST respond in less than 10 seconds, either with a successful response, a timeout, or an error message.		Mandatory (Phase 1)
Rest-NF-02	The Catalogue MUST block clients after a maximum of 10 incorrect authentication attempts or requests with incorrect identity claims.	Execute ten incorrect authentication attempts. The eleventh attempt must not return a response message.	Optional (Phase 2)
Rest-NF-03	A Catalogue SHOULD store blocked users in a blacklist.		Optional (Phase 2)
Rest-NF-04	The Catalogue SHOULD reduce the rate limit of a user when he sends more than 10 requests per second to avoid denial of services.		Optional (Phase 2)
Rest-NF-05	The full REST API MUST be documented in a well-structured format e.g., OpenAPI 3.0 standard.	Followed best practices and tool support.	Mandatory (Phase 1)

Rest-NF-06	The entire REST API MUST support TLS encryption with at least version 1.3 as defined by RFC 8446.		Mandatory (Phase 1)
Rest-NF-07	Each API Call MUST have an operation identifier.		Mandatory (Phase 1)

***Table 36**: Catalogue REST API Subsystem Non-Functional Requirements*

Appendix A: Self-Description Core Ontology

The following non-normative listing shows the Self-Description Core Ontology, represented as an RDF graph in Turtle serialization.

```
@prefix owl:    <http://www.w3.org/2002/07/owl#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix skos:   <http://www.w3.org/2004/02/skos/core#> .
@prefix cc: <http://creativecommons.org/ns#> .
@prefix voaf: <http://purl.org/vocommons/voaf#> .

@prefix gax:    <http://w3id.org/gaia-x/core#> .

gax: a voaf:Vocabulary, owl:Ontology ;
    rdfs:label "Gaia-X Ontology"@en ;
    cc:license <http://www.apache.org/licenses/LICENSE-2.0> ;
    dct:creator "Gaia-X Open Work Package 'Self-Description'" ;
    dct:created "2020-07-06T12:00:00+01:00"^^xsd:dateTimeStamp ;
    dct:modified "2020-04-13T12:00:00+01:00"^^xsd:dateTimeStamp ;
    owl:versionInfo "0.1" ;
    vann:preferredNamespaceUri    "http://w3id.org/gaia-x/core#" ;
    vann:preferredNamespacePrefix "gax" ;
    void:vocabulary vann:, void:, voaf:, dct:, foaf: .

#############
# Core Classes from the Conceptual Model
#############
```

```
gax:Participant
    a owl:Class ;
    rdfs:label "Participant"@en ;
    rdfs:comment "A Participant is a natural or legal person who is
identified, authorized and has a Gaia-X Self-Description."@en ;
.

gax:Provider
    a owl:Class ;
    rdfs:subClassOf gax:Participant ;
    rdfs:label "Provider"@en ;
    rdfs:comment "A Participant who provides Resources in the Gaia-X
ecosystem."@en ;
    rdfs:subClassOf [ a owl:Restriction ;
          owl:onProperty gax:provides ;
          owl:minCardinality 1 ;
    ] ;
    rdfs:subClassOf [ a owl:Restriction ;
          owl:onProperty gax:providesResourcesFrom ;
          owl:minCardinality 1 ;
    ] ;
    rdfs:subClassOf [ a owl:Restriction ;
          owl:onProperty gax:owns ;
          owl:minCardinality 1 ;
    ] ;
    rdfs:subClassOf [ a owl:Restriction ;
          owl:onProperty gax:operates ;
          owl:minCardinality 1 ;
    ] ;
.

gax:Federator
    a owl:Class ;
    rdfs:subClassOf gax:Participant ;
    rdfs:label "Federator"@en ;
    rdfs:comment "A Federator is a Participant who enables a Federation
Service."@en ;
    rdfs:subClassOf [ a owl:Restriction ;
          owl:onProperty gax:provides ;
          owl:minCardinality 1 ;
    ] ;
.

gax:Consumer
    a owl:Class ;
    rdfs:subClassOf gax:Participant ;
    rdfs:label "Consumer"@en ;
    rdfs:comment "A Participant who consumes and leverages Service Instance
in the Gaia-X ecosystem to enable digital offerings for End Users."@en ;
    rdfs:subClassOf [ a owl:Restriction ;
          owl:onProperty gax:consumes ;
          owl:minCardinality 1 ;
    ] ;
.

gax:FederationService
    a owl:Class ;
    rdfs:label "Federation Service"@en ;
    rdfs:comment "Federation Services provide the foundation for the
operational implementation of the Gaia-X model. An Open-Source community-
based reference implementation of them will be provided by the Federation
Services projects under specification and oversight by Gaia-X AISBL."@en
;
    rdfs:subClassOf [ a owl:Restriction ;
          owl:onProperty gax:definesSchemas ;
          owl:minCardinality 1 ;
    ] ;
.

gax:ServiceOffering
```

```
    a owl:Class ;
    rdfs:label "Service Offering"@en ;
    rdfs:comment "A Service Offering is a set of Assets and Resources, which a Provider bundles into an offering."@en .

gax:Resource
    a owl:Class ;
    rdfs:label "Resource"@en ;
    rdfs:comment "Behavior element used by the Service Instance via the Service Offering composition."@en .

gax:Asset
    a owl:Class ;
    rdfs:label "Asset"@en ;
    rdfs:comment "Static structural element, used to compose the Service Offering."@en .

#############
# Core Classes from the Conceptual Model (without own SDs)
#############


gax:AssetOwner
    a owl:Class ;
    rdfs:label "Asset Owner"@en ;
    rdfs:comment "A natural or legal person who is in legal possession of the Asset."@en ;
    rdfs:subClassOf [ a owl:Restriction ;
          owl:onProperty gax:owns ;
          owl:minCardinality 1 ;
    ] ;
    rdfs:subClassOf [ a owl:Restriction ;
          owl:onProperty gax:legallyEnablesResourceProvision ;
          owl:minCardinality 1 ;
    ] ;
```

```
.

gax:EndUser
    a owl:Class ;
    rdfs:label "End User"@en ;
    rdfs:comment "A natural person not being Principal, using digital
offering from a Consumer. End-Users own an identity within the Consumer
context."@en .

gax:ServiceInstance
    a owl:Class ;
    rdfs:label "Service Instance"@en ;
    rdfs:comment "Realisation by the Provider of the Service Offering."@en
.

gax:Contract
    a owl:Class ;
    rdfs:label "Contract"@en ;
    rdfs:comment "Contract means the binding legal agreement describing a
Service Instance and includes all rights and obligations."@en .

#############
# Extended Classes from the Conceptual Model
#############

gax:DataAsset
    a owl:Class ;
    rdfs:subClassOf gax:Asset ;
    rdfs:label "Data Asset"@en ;
    rdfs:comment "Data Asset is a subclass of Asset and consist of data in
any form and necessary information for data sharing."@en .

gax:Interconnection
    a owl:Class ;
    rdfs:subClassOf gax:Asset ;
    rdfs:label "Interconnection"@en ;
    rdfs:comment "Interconnection is a dedicated category of Assets. An Interconnection is a connection between two or multiple nodes. These nodes are usually located at different locations and owned by different stakeholders, such as customers and/or providers. The Interconnection between the nodes can be seen as a path, which exhibits special characteristics, such as latency and bandwidth guarantees, that go beyond the characteristics of a path over the public Internet."@en .


gax:Node
    a owl:Class ;
    rdfs:subClassOf gax:Asset ;
    rdfs:label "Node"@en ;
    rdfs:comment "A Node is a sub class of Assets. A Node represents a computational or physical entity that hosts, manipulates, or interacts with other computational or physical resources."@en .


gax:SoftwareAsset
    a owl:Class ;
    rdfs:subClassOf gax:Asset ;
    rdfs:label "Software Asset"@en ;
    rdfs:comment "Software Assets are a form of Assets that consist of non-physical functions."@en .



#############
# Properties
#############


gax:providesResourcesFrom
    a            owl:ObjectProperty ;
    rdfs:label   "provides resources from"@en ;
    rdfs:domain  gax:Provider ;
    rdfs:range   gax:AssetOwner .


gax:legallyEnablesResourceProvision
    a            owl:ObjectProperty ;
    rdfs:label   "legally enables resource provision"@en ;
    rdfs:domain  gax:AssetOwner ;
    rdfs:range   gax:Provider .

gax:owns
    a            owl:ObjectProperty ;
    rdfs:label   "owns"@en ;
    rdfs:domain [ owl:unionOf (gax:Provider gax:AssetOwner)] ;
    rdfs:range gax:Asset .

gax:operates
    a            owl:ObjectProperty ;
    rdfs:label   "operates"@en ;
    rdfs:domain  gax:Provider ;
    rdfs:range   gax:Resource .

gax:provides
    a            owl:ObjectProperty ;
    rdfs:label   "provides"@en ;
    rdfs:domain [ owl:unionOf (gax:Provider gax:Federator)] ;
    rdfs:range [ owl:unionOf (gax:ServiceInstance gax:FederationService)]
.

gax:definesSchemas
    a            owl:ObjectProperty ;
    rdfs:label   "defines schemas"@en ;
    rdfs:domain  gax:FederationService ;
    rdfs:range   gax:ServiceOffering .

gax:usesAndConforms
    a            owl:ObjectProperty ;
    rdfs:label   "uses and conforms"@en ;
    rdfs:domain  gax:ServiceOffering ;
    rdfs:range   gax:FederationService .

gax:composes
```

```
    a             owl:ObjectProperty ;
    rdfs:label    "composes"@en ;
    rdfs:domain   gax:ServiceOffering ;
    rdfs:range [ owl:unionOf (gax:Resource gax:Asset)] .

gax:realizesBy
    a             owl:ObjectProperty ;
    rdfs:label    "realized by"@en ;
    rdfs:domain   gax:ServiceOffering ;
    rdfs:range    gax:ServiceInstance .

gax:offersTo
    a             owl:ObjectProperty ;
    rdfs:label    "offers to"@en ;
    rdfs:domain   gax:Consumer ;
    rdfs:range    gax:EndUser .

gax:usesDigitalOfferingBy
    a             owl:ObjectProperty ;
    rdfs:label    "uses digital offering by"@en ;
    rdfs:domain   gax:EndUser ;
    rdfs:range    gax:Consumer .

gax:consumes
    a             owl:ObjectProperty ;
    rdfs:label    "consumes"@en ;
    rdfs:domain   gax:Consumer ;
    rdfs:range    gax:ServiceInstance .

gax:uses
    a             owl:ObjectProperty ;
    rdfs:label    "uses"@en ;
    rdfs:domain   gax:EndUser ;
    rdfs:range    gax:ServiceInstance .
```

```
gax:reliesOn
    a             owl:ObjectProperty ;
    rdfs:label    "relies on"@en ;
    rdfs:domain   gax:ServiceInstance ;
    rdfs:range    gax:Contract .

gax:managedBy
    a                    owl:ObjectProperty  ;
    rdfs:label           "managed by"@en ;
    rdfs:comment           "Declares a Gaia-X participant that manages / 
maintains this asset."@en ;
    skos:note              "Should be used to link Gaia-X asset to their 
respective Participant, as they cannot exist without one."@en ;
    rdfs:domain          gax:Asset ;
    rdfs:range           gax:Participant .

gax:providedBy
    a             owl:ObjectProperty ;
    rdfs:label    "provided by"@en ;
    rdfs:domain    [ owl:unionOf (gax:Node gax:Service)] ;
    rdfs:range    gax:Provider .

gax:ownedBy
    a             owl:ObjectProperty ;
    rdfs:label    "owned by"@en ;
    rdfs:range    gax:Participant .

#################
# Meta Ontology #
#################

gax:MustCriterion
    rdfs:subClassOf rdf:Property ;
    rdfs:label      "must criterion"@en ;
    rdfs:comment    "a property for which a value must be provided"@en ;
.

gax:OverridableCriterion
    rdfs:subClassOf rdf:Property ;
    rdfs:label      "overridable criterion"@en ;
    rdfs:comment    "a property whose value can be inherited, but may be
overridden"@en ;
.

gax:HiddenCriterion
    rdfs:subClassOf rdf:Property ;
    rdfs:label      "hidden criterion"@en ;
    rdfs:comment    "a property that should be hidden when generating
visualizations of an entity"@en ;
.
```

Appendix B: REST API

```
# Origin: https://gitlab.com/gaia-x/gaia-x-technical-committee/federation-
services/wp2/catalogue-specification-artefacts/-/blob/master/rest-api.yaml

openapi: 3.0.1
info:
  title: Gaia-X Catalogue
  description: 'This is the REST API of the Gaia-X catalogue.'
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.0
servers:
- url: https://api.gaiax.io/v1

# Authentication by OAuth2 (no scoping at this point)
components:
  securitySchemes:
    oAuthNoScopes:
      type: oauth2
      description: This API uses OAuth 2 with the implicit grant flow. [More info](https://api.example.com/docs/auth)
      flows:
        implicit:
          authorizationUrl: https://api.gaiax.io/oauth2/authorize
          scopes:
            read_self-descriptions: read the Self-Descriptions
            write_self-descriptions: Add, delete, and update a Self-Description
            query: Send queries
  responses:
    NotFound:
      description: The specified resource was not found
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
    Unauthorized:
      description: Unauthorized
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
    ServerError:
      description: May contain hints how to solve the error or indicate what went wrong at the server. Must not outline any information about the internal structure of the server.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
    ClientError:
      description: May contain hints how to solve the error or indicate what was wrong in the request.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
  schemas:
    # Schema for error response body
    Error:
      type: object
      properties:
        code:
          type: string
        message:
          type: string
      required:
        - code
        - message

    Statements:
      type: object
      properties:
        statements:
          type: array
          items:
            $ref: '#/components/schemas/Statement'
      maxItems: 1 # only one statement can be sent for now. The array is for
future
      # extensions

    Statement:
      type: object
      properties:
        statement:
          type: string
          example: 'Match (m:Movie) where m.released > 2000 RETURN m'
        parameters:
          $ref: '#/components/schemas/Parameters'
      required:
      - statement

    Parameters:
      type: object
      properties:
        limit:
          type: string
          example: '10'
        offset:
          type: string
          example: '5'

    Results:
      type: object
      properties:
        results:
          type: array
          items:
            $ref: '#/components/schemas/Result'
          maxItems: 1
      required:
      - results

    Result:
      type: object
      properties:
        columns:
          type: array
          example: ["column name 1", "column name 2"]
          items:
            type: string
            example: 'column names'
        data:
          type: array
          items:
            $ref: '#/components/schemas/Data'
      required:
      - columns
      - data
    VerificationResult:
      type: object
      properties:
        verification-timestamp:
          type: string
        lifecycle-status:
          type: string # does the self-description have a lifecycle status in this
catalogue?
        issuer:
          type: object # contains the id of the issuers self-description if
available
        issued-date:
          type: string
        signatures:
          type: array
          items:
            type: object
      required:
      - verification-timestamp
      - lifecycle-status
      - issuer
      - issued-date
      - signatures

    Data:
      type: object
      properties:
        row:
          type: array
          example: ["value1", "value2"]
          items:
            type: string
        meta:
          type: array
          example: ["meta1", "meta2"]
          items:
            type: string
      required:
      - row
      - meta

    Participant:
      type: object
      properties:
        id:
          type: string
          description: Global ID of the participant
        name:
          type: string
        public-key:
          type: string
        self-description:
          type: string

    User:
      type: object
      properties:
        id:
          type: string
          description: Internal catalogue user id
          example: 'ExampleCompany-John-Doe'
        participantId:
          type: string
          description: Global ID of the associated participant
          example: 'ExampleCompany'
        username:
          type: string
          example: 'John Doe'
        email:
          type: string
        roleIds:
          type: array
          items:
            $ref: '#/components/schemas/Role'
    Role:
      type: object
      properties:
        id:
          type: string
          example: 'Ro-MU-CA'

    Session:
      type: object
      properties:
        userId:
          type: string
        creationDate:
          type: string
        status:
          type: string
        roleIds:
          type: array
          items:
            $ref: '#/components/schemas/Role'

    Self-Description:
      type: object
      properties:
        sd-hash:
          type: string
        status:
          type: integer
        issuer:
          type: string
        issue-date:
          type: string
        status-date:
          type: string
          description: The last time the status changed (for this Catalogue)
```

```
security:
  - oAuthNoScopes:
    - read_self-descriptions
    - write_self-descriptions
    - query

tags:
- name: discovery
  description: 'Announce the endpoints of this Catalogue.'

- name: self-descriptions
  description: 'Retrieving Self-Descriptions from the Catalogue. All Self-
Descriptions are JSON-LD files. They are referenced by their sha256 hash.
Catalogues synchronize by downloading changesets (lists of hashes) from known
other Catalogues and reading the full Self-Descriptions of entries that are unknown
to them.'
  externalDocs:
    description: Find out more
    url: http://gaiax.io

- name: sandbox
  description: 'Try changes to the JSON-LD Self-Descriptions against the catalogue
in a sandbox, i.e. the changes are not really applied. But the error messages
allow the debugging of Self-Descriptions wrt trust and validation of the content
on a syntactical and semantic level.'
  externalDocs:
    description: Find out more
    url: http://gaiax.io

- name: query
  description: 'Send graph queries to this Catalogue.'

- name: users
  description: User management for Catalogues not connected to an external IAM
system

- name: participants
  description: Management for registered participants in the catalogue
```

```
- name: schemas
  description: The format of the self-descriptions are defined by schemas in the catalogue. Here you get information about the latest schema.

- name: roles
  description: Management for the permission roles in the catalogue

- name: verification
  description: The Catalogue provides a verification service for e.g. checking the syntax

paths:
  /:
    get:
      tags:
      - discovery
      summary: 'Announce all endpoints'
      operationId: discovery
      responses:
        200:
          description: 'Provides a JSON element with relative paths to all other
            available endpoints of this Catalogue.'
          content:
            application/json:
              schema:
                type: object
                properties:
                  query:
                    type: string
                    example: './query'
                  self-descriptions:
                    type: string
                    example: './self-descriptions'
                  self_description_hash:
                    type: string
                    example: './self-descriptions/{self_description_hash}'
            application/ld+json:
              schema:
                type: object
                properties:
                  '@context':
                    type: object
                    properties:
                      gax:
                        type: string
                        example: "http://gaia-x.eu/gaiaxOntology#"
                  '@id':
                    type: string
                    example: "<the identifier of this catalogue>"
                  gax:hasQueryEndpoint:
                    type: string
                    example: './query'
                  gax:hasSelfDescriptionEndpoint:
                    type: string
                    example: './self-descriptions'
                  gax:hasSelfDescriptionHashEndpoint:
                    type: string
                    example: './self-descriptions/{self_description_hash}'
                  gax:isOperatedBy:
                    type: string
                    example: 'http://example.org/CatalogueProvider'

  /verifications/self-descriptions:
    get:
      tags:
        - verification
      summary: Show a HTML page to verify (portions of) a signed Self-Description
      operationId: verifyPage
      responses:
        200:
          description: 'HTML document that contains a query field to verify
(portions of) Self-Descriptions.'
          content:
            text/html:
              schema:
                type: string
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'

    post:
      tags:
      - verification
      summary: 'Send a JSON-LD document to verify with the information from the
Catalogue'
      operationId: verify
      requestBody:
        description: 'JSON-LD document to be verified object to send queries. Use
"application/json" for openCypher queries. A Catalogue may also support the other
content types depending on its supported query languages but only
"application/json" is mandatory.'
        content: {}

      responses:
        200:
          description: 'Verification result'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/VerificationResult'
        400:
          $ref: '#/components/responses/ClientError'
        408:
          description: 'Query Timeout: the query took longer than the configured
timeout interval. The client needs to rewrite the query so it can be processed
faster.'
        500:
          $ref: '#/components/responses/ServerError'

  /sessions:
    get:
      tags:
        - session
      responses:
        200:
          description: Get information on the current session
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Session'

  /sessions/logout:

    get:
      tags:
        - session
      responses:
        200:
          description: The current session was closed

  /participants:
    get:
      tags:
      - participants
      summary: Get the registered participants
      operationId: listParticipants
      responses:
        200:
          description: List of registered participants
          content:
            application/json:
              schema:
                 $ref: '#/components/schemas/Participant'
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'

    post:
      tags:
      - participants
      summary: Register a new participant in the catalogue
      operationId: addParticipant
      requestBody:
        description: Participant Self-Description
        content:
          application/json-ld:
            schema:
                $ref: '#/components/schemas/Participant'

      responses:
        201:
          description: Created
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'

  /participants/{participantId}:
    get:
      tags:
      - participants
      summary: Get the registered participant
      operationId: getParticipant
      parameters:
        - in: path
          name: participantId
          required: true
          description: The participantId to get.
          schema:
            type: string

      responses:
```

```
      200:
        description: The requested participant
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Participant'
      400:
        $ref: '#/components/responses/ClientError'
      500:
        $ref: '#/components/responses/ServerError'

  put:
    tags:
    - participants
    summary: Update a participant in the catalogue
    operationId: updateParticipant
    parameters:
      - in: path
        name: participantId
        required: true
        description: The participant to update.
        schema:
          type: string
    responses:
      200:
        description: Updated Participant
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Participant'
      400:
        $ref: '#/components/responses/ClientError'
      500:
        $ref: '#/components/responses/ServerError'



  delete:
    tags:
    - participants
    summary: Delete a participant in the catalogue
    operationId: deleteParticipant
    parameters:
      - in: path
        name: participantId
        required: true
        description: The participant to delete.
        schema:
          type: string
    responses:
      200:
        description: Deleted Participant
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Participant'
      400:
        $ref: '#/components/responses/ClientError'
      500:
        $ref: '#/components/responses/ServerError'

/participants/{participantId}/users:
  get:
    tags:
    - participants
    summary: Get all users of the registered participant
    operationId: getUsersOfParticipant
    parameters:
      - in: path
        name: participantId
        required: true
        description: The participant to create.
        schema:
          type: string
    responses:
      200:
        description: Users of the participant
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/User'
      400:
        $ref: '#/components/responses/ClientError'
      500:
        $ref: '#/components/responses/ServerError'

/users:

  get:
    tags:
    - users
    summary: List the registered users
    operationId: listUsers
    responses:
      200:
        description: List of usernames
        content:
          application/json:
            schema:
              type: array
              items:
                 $ref: '#/components/schemas/User'
      400:
        $ref: '#/components/responses/ClientError'
      500:
        $ref: '#/components/responses/ServerError'

  post:
    tags:
    - users
    summary: Register a new user to the associated participant in the catalogue
    operationId: addUser
    requestBody:
      description: User Self-Description
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/User'

    responses:
      201:
        description: Created
      400:
        $ref: '#/components/responses/ClientError'
      500:
        $ref: '#/components/responses/ServerError'

/users/{userId}:

  get:
    tags:
    - users
    summary: Get the user profile
    operationId: getUser
    parameters:
    - name: userId
      in: path
      required: true
      schema:
        type: string
    responses:
      200:
        description: User profile
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'
      400:
        $ref: '#/components/responses/ClientError'
      500:
        $ref: '#/components/responses/ServerError'

  put:
    tags:
    - users
    summary: Update the user profile
    operationId: updateUser
    parameters:
    - name: userId
      in: path
      required: true
      schema:
        type: string
    responses:
      200:
        description: Updated user profile
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'
      400:
        $ref: '#/components/responses/ClientError'
      500:
        $ref: '#/components/responses/ServerError'

  delete:
    tags:
    - users
    summary: Delete a user
    operationId: deleteUser
    parameters:
    - name: userId
      in: path
      required: true
```

```
        schema:
          type: string
      responses:
        200:
          description: Deleted user profile
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'


  /users/{userId}/roles:

    get:
      tags:
      - users
      summary: Get the roles of the user
      operationId: getUserRoles
      parameters:
      - name: userId
        in: path
        required: true
        schema:
          type: string
      responses:
        200:
          description: User roles
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Role'
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'

    put:
      tags:
      - users
      summary: Update the roles of the user
      operationId: updateUserRoles
      parameters:
      - name: userId
        in: path
        required: true
        schema:
          type: string
      requestBody:
        description: List of roles which should be assigned to the user
        content:
            application/json:
              schema:
                $ref: '#/components/schemas/Role'
      responses:
        200:
          description: All assigned roles of the user
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Role'
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'

  /queries:

    get:
      tags:
      - query
```

```
      summary: Retrieve an HTML website to send openCypher queries to the Catalogue
      operationId: querywebsite
      security:
        - oAuthNoScopes:
          - query
      responses:
        200:
          description: 'HTML document that contains a query field for openCypher queries.'
          content:
            text/html:
              schema:
                type: string
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'

    post:
      tags:
      - query
      summary: 'Send a query to the Catalogue'
      operationId: query
      parameters:
      - in: header
        name: query-language
        schema:
          type: string
          enum: [openCypher, application/sparql-query, sparql*]
          default: openCypher
        required: true
      security:
        - oAuthNoScopes:
          - query
      requestBody:
        description: 'JSON object to send queries. Use "application/json" for openCypher queries. A Catalogue may also support the other content types depending on its supported query languages but only "application/json" is mandatory.'
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Statements'
          application/sparql-query:
            example: ''
          sparql*:
            example: ''

      responses:
        200:
          description: 'successful query'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Results'
            application/sparql-results+xml:
              example: ''
            text/turtle:
              example: ''
            text/html:
              example: ''
        400:
          description: 'Malformed Message: The receveived request cannot be
processed, either because its syntax is incorrect or forbidden query clauses are
used. For instance, it is not allowed to manipulate data through the query
endpoint.'
          content:
            application/json:
              schema:
                $ref: '#/components/responses/ClientError'
        408:
          description: 'Query Timeout: the query took longer than the configured
timeout interval. The client needs to rewrite the query so it can be processed
faster.'
        500:
                $ref: '#/components/responses/ServerError'
```

```
  /self-descriptions:

    get:
      tags:
      - self-descriptions
      summary: Get the list of Self-Descriptions in the Catalogue
      operationId: readSelfDescriptions
      parameters:
      - name: daterange
        in: query
        description: Range of dates for the Self-Descriptions (when the SD was
first known this Catalogue)
        required: false
        schema:
          type: string
      - name: issuer
        in: query
        description: Filter for the issuer of the Self-Description. This is the
unique ID of the Participant that has prepared the Self-Description.
        required: false
        schema:
          type: string
      - name: offset
        in: query
        schema:
            type: integer
            minimum: 0
            default: 0
        required: false
        description: The number of items to skip before starting to collect the
result set.
      - in: query
        name: limit
        schema:
            type: integer
            minimum: 1
            maximum: 1000
          default: 100
        required: false
        description: The number of items to return.
      responses:
        200:
          description: List of self-description in JSON-LD format
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Self-Description'



        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'

    post:
      tags:
      - self-descriptions
      summary: Add a new self-description to the catalogue
      operationId: addSelfDescription
      security:
        - oAuthNoScopes:
          - write_self-descriptions
      requestBody:
        description: The new Self-Description
        content:
          application/json: {}
        required: true
      responses:
        201:
          description: Created
        405:
          description: Invalid input
          content: {}
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'

  /self-descriptions/{self_description_hash}:

    get:
      tags:
      - self-descriptions
      summary: Read a Self-Description by its hash
      description: Returns a single Self-Description
      operationId: readSelfDescriptionByHash
      parameters:
      - name: self_description_hash
        in: path
        description: Hash of the self-description
        required: true
        schema:
          type: string
      security:
        - oAuthNoScopes:
          - read_self-descriptions
      responses:
        200:
          description: The requested Self-Description
          content:
            application/json: {}
        404:
          description: Self-Description not found
          content: {}
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'
    put:
      tags:
      - self-descriptions
      summary: Change the lifecycle state of a self-description
      operationId: updateSelfDescription
      parameters:
      - name: self_description_hash
        in: path
        description: Hash of the self-description
        required: true
        schema:
          type: string
      requestBody:
        description: Lifecycle update message
        content:
          application/json: {}
        required: true
      responses:
        405:
          description: Invalid input
          content: {}
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'

    delete:
      tags:
      - self-descriptions
      summary: Delete a self-description
      operationId: deleteSelfDescription
      parameters:
      - name: self_description_hash
        in: path
        description: Hash of the self-description
        required: true
        schema:
          type: string
      responses:
        200:
          description: OK
        405:
          description: Invalid input
          content: {}
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'

  /schemas/latest:

    get:
      tags:
      - schemas
      summary: Get the latest schema of all types
      operationId: getLatestSchemas

      responses:
        200:
          description: The latest schemas of all types
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
        405:
          description: Invalid input
          content: {}
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'
  /schemas/latest/{type}:

    get:
      tags:
      - schemas
      summary: Get latest schemas of a specific type
      operationId: getLatestSchemaOfType
      parameters:
      - name: type
        in: path
        description: Type of the requested Self-Description schema e.g. Service
        required: true
        schema:
          type: string
      responses:
        200:
          description: The latest schema of requested types
          content:
            application/json: {}
        405:
          description: Invalid input
          content: {}
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'

  /roles:

    get:
      tags:
      - roles
      summary: Get all possible roles in the catalogue
      operationId: getAllRoles
      responses:
        200:
          description: All roles
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Role'
        400:
          $ref: '#/components/responses/ClientError'
        500:
          $ref: '#/components/responses/ServerError'
```

Appendix C: Database Structure for Self-Description Metadata

```
-- The following is the SQL definition of a database schema for metadata
-- to the raw JSON-LD Self-Descriptions

CREATE TABLE statuscodes (
    id             INTEGER        PRIMARY KEY,
    status         VARCHAR(20)    NOT NULL UNIQUE
);

INSERT INTO statuscodes (id, status)
VALUES
  (1, 'active'),
  (2, 'deprecated'),  -- deprecated: a new version of the self-description
                      --             exists
  (3, 'eol'),         -- eol: end of lifetime (assets can continue to run,
                      --      but no more provisioning)
  (4, 'revoked');     -- revoked: any of the signing parties or a trusted
                      --          party revoked

CREATE TABLE self_descriptions (
    hash           CHAR(64)       PRIMARY KEY, -- Hash of the JSON-LD
                                               -- self-description
    subject        VARCHAR        not null,    -- GAIA-X identifier of the
                                               -- subject (@id) of the JSON-LD
                                               -- self-description
    validators     VARCHAR[],                  -- GAIA-X identifiers of the
                                               -- participants who validated the
                                               -- self-description
    issued         DATE           not null,    -- When was the self-description
                                               -- issued?
    received       DATE           not null,    -- When was the self-description
                                               -- first received by this
                                               -- catalogue?
    jsonld         VARCHAR        not null,    -- The raw JSON-LD matching
                                               -- the hash
    status         INTEGER        NOT NULL DEFAULT 1, -- Status of the
                                                      -- Self-Description
CONSTRAINT fk_statuscodes FOREIGN KEY (status) references statuscodes (id)
                                               -- Only valid status are allowed
);
```

Appendix D: Gaia-X Architecture Decision Records

```
ADR-001: JSON-LD as the Exchange Format for Self-Descriptions
=============================================================

:adr-id: 001
:revnumber: 1.0
:revdate: 06-07-2020
:status: accepted
:author: Self-Description WP, Catalogue WP
:stakeholder: Self-Description WP, Catalogue WP

Summary
-------

GAIA-X needs to define a serialization format for the exchange of
Self-Descriptions. The exchange format allows the self-descriptions to be
serialized into files for transportation between Services and Catalogues.
JSON-LD is selected as the Self-Description serialization format.

The following is a minimal example for the JSON-LD format::

    {
        "@context": "https://json-ld.org/contexts/gaia-x.jsonld",
        "@id": "http://dbpedia.org/resource/MyService",
        "provider": "http://dbpedia.org/resource/MyProvider",
        "name": "MyService"
    }

Linked Data Proofs 1.0 (https://w3c-ccg.github.io/ld-proofs/, currently in draft
status).

An example for a signed document with Linked Data Proofs is this::

    {
      "@context": "https://www.w3.org/2018/credentials/examples/v1",
      "title": "Hello World!",
      "proof": {
        "type": "Ed25519Signature2018",
        "proofPurpose": "assertionMethod",
        "created": "2019-08-23T20:21:34Z",
        "verificationMethod": "did:example:123456#key1",
        "domain": "example.org",
        "jws": "eyJ0eXAiOiJK...gFWFOEjXk"
      }
    }

Context
-------

The Architecture Document (June 2020) states that Self-Descriptions are
expressed in an extensible format.

JSON is an established data serialization format. It is both human-readable and
machine-interpretable.

JSON-LD combines JSON with semantic technologies (ontologies) from the Linked
Data community. Specifically, the RDF-Standard is referenced. So the
schema-definitions (and tooling) from the established RDF format can be reused.
```

```
Alternative Technologies
~~~~~~~~~~~~~~~~~~~~~~~~

Decision Statements
-------------------

The serialization format for the exchange of GAIA-X Self-Descriptions is JSON-LD
1.1.

The RDF 1.1-standard is used to express an extensible hierarchy of schemas for
Self-Descriptions with well-known attributes.

Cryptographic signatures are added to Self-Descriptions according to the Linked
Data Proofs 1.0 specification.

Consequences
------------

All Self-Descriptions need to be ready for serialization into the JSON-LD
format.

The extensible hierarchy of self-description attributes is expressed in an
ontology according to the RDF standard.

The Identifier of GAIA-X Assets and Participants must be IRIs (Internationalized
Resource Identifiers [RFC3987]) so that they can be used for cross-referencing
between Self-Descriptions in the JSON-LD format.

ADR References
--------------

External References
-------------------

* [JSON-LD] JSON-LD 1.1 - A JSON-based Serialization for Linked Data,
  https://www.w3.org/TR/json-ld11
* [LDP] Linked Data Proofs 1.0, https://w3c-ccg.github.io/ld-proofs/
* [RDF] RDF 1.1 Concepts and Abstract Syntax,
  https://www.w3.org/TR/rdf11-concepts/


ADR-002: REST as the Interface Technology for Federation Services
=================================================================

:adr-id: 002
:revnumber: 0.9
:revdate: 23-07-2020
:status: accepted
:author: Catalogue WP
:stakeholder: All Federation Services

Summary
-------

GAIA-X defines a set of Federation Services. These Services provide an API for
the development of client applications. This "client" could be an application
developed by a GAIA-X Participant or a user interface that is part of GAIA-X
itself.

For the internal consistency of GAIA-X and the Federation Services, a common
technology and design principles should be selected for the Federation Service's
```

API interface.

The ADR proposes REST (HTTP+JSON) as the interface technology and OpenAPI as the interface definition language.

Discussion

Many different protocols are in use for internet-based software interfaces. Following is an (incomplete) overview listing only the most common technologies.

- SOAP Webservices (https://www.w3.org/TR/soap/)
- XML-RPC (http://xmlrpc.com/)
- REST (HTTP+JSON)
- Object-Oriented Interfaces (Corba, OPC UA, ...)
- Message-Oriented Interfaces (e.g. via Kafka, MQTT, AMQP, DDS, ...)

In principle, the Federation Services could be implemented using either of the mentioned interface technologies. But there are further criteria besides the core functionality to take into consideration. Selecting a widely used technology with a proven track-record ensures that GAIA-X Participants have developers with the skills to immediately make use of the Federation Services. Furthermore, the established technologies are more likely to be widely supported by the different programming languages and environments in the long-term.

From those criteria, the choice has fallen on a combination of REST (HTTP+JSON). It is the mostly widely used technology for web-API development (also used by the current hyperscalers) and has mature tooling.

Based on the choice of REST (HTTP+JSON), several competing formats exist for expressing the interface definitions in a human and machine-readable format.

- OpenAPI (http://spec.openapis.org/oas/v3.0.3)
- RAML (https://raml.org/)
- Hydra (https://www.hydra-cg.com/)
- Custom Format for GAIA-X

Again, all these technologies could perform the task to a sufficient degree. We select the most widely used alternative OpenAPI.

Decision Statements

The GAIA-X Federation Services use JSON-Documents transferred via a HTTP/REST API for their interfaces.

Special use cases (e.g. for streaming data) might receive an exemption to use different interface technologies.

The interfaces of the Federation Services are specified in the OpenAPI Specification (v3 or later). http://spec.openapis.org/oas/v3.0.3

Consequences

Developers can reuse tools for the commonly used combination of HTTP+JSON for the Federation Services.

Code stubs to interact with the Federation Services can be auto-generated from the OpenAPI definitions for many programming environments.

```
The Work-Packages that define the interfaces of the Federation Services use
OpenAPI to define the interfaces in a human and machine-readable format.

Notes
-----

The REST API specifications are described in https://gitlab.com/gaia-x/gaia-x-
core/gaia-x-core-document-technical-concept-architecture/-
/blob/master/architecture_document/federation_services.rst

ADR References
--------------

* ADR-001: JSON-LD as the Exchange Format for Self-Descriptions

External References
-------------------

* [REST] https://en.wikipedia.org/wiki/Representational_state_transfer
* [OpenAPI] http://spec.openapis.org/oas/v3.0.3


ADR-003: Lifecycle of Self-Descriptions
=======================================

:adr-id: XXX
:revnumber: 1.0
:revdate: 05-11-2020
:status: proposed
:author: Catalogue WP
:stakeholder: Self-Description WP, Catalogue WP

Summary
-------

There are four possible states for the lifecycle of GAIA-X Self-Descriptions:
active, eol (end of life), deprecated and revoked.

Context
-------

GAIA-X Assets (Nodes, Services, etc.) and their Self-Descriptions can be updated
over time. Furthermore, there can be Self-Descriptions that are abandoned or
even revoked for containing false information. This needs to be explicitly
tracked to prevent the Catalogue and GAIA-X participants to work with outdated
Self-Descriptions.

The claims of the Self-Descriptions in JSON-LD format can carry cryptographic
proofs based on hash signatures. Furthermore, the hash of the JSON-LD file can
be used as an identifier to reference to the source of information from the
Catalogue. Hence the JSON-LD files are immutable and can only be replaced as a
whole. The state of the Self-Descriptions therefore needs to be tracked as
metadata outside of the JSON-LD files itself.

End of Life: The Self-Descriptions are the source information for the GAIA-X
Catalogue. In order for the Catalogue to contain only up-to-date information it
should be "self-cleaning" whereby outdated information is removed automatically.
This can be achieved by timeout dates attached to every Self-Description after
which they are end-of-life. It is recommended that the automatic timeout date of
Self-Descriptions is set rather low, e.g. 90 days. This has proven useful in the
context of TLS certificates [LetsEncrypt] where a frequent renewal forces
```

providers that automated update systems are put in place instead of infrequent manual updates.

Deprecated: If two versions of GAIA-X Assets (Nodes, Services, Data etc.) are offered at the same time, then they each have independent Self-Descriptions. In order for the Self-Descriptions to be self-contained, there shall be no partial updates to Self-Descriptions. The entire JSON-LD file is published in an updated version and the old Self-Description is deprecated with reference to the updated Self-Description.

Revoked: GAIA-X needs to protect against bad actors that might not be able or willing to correct false information. Hence a revocation mechanism is put into place by which Self-Descriptions can be marked as non-active. Revocation can be performed by the original issuer of a Self-Description and also by trusted parties.

Decision Statements

The Self-Descriptions in JSON-LD format (ADR-001) have an additional state information that can change over time. The possible states are:

- active
- eol (end of life after a timeout date)
- deprecated (by a newer Self-Description)
- revoked (by a trusted party)

The default state is "active". The other states are terminal, so no further state transitions are made once a Self-Description has become non-active.

Non-active Self-Descriptions might still be available in the JSON-LD format in the historical record. But they are no longer considered for new search queries in the Catalogue, etc.

Self-Descriptions have a timeout date after which they are in the "eol" state. The timeout date is part of the JSON-LD file and considered in cryptographic signatures.

Self-Descriptions in the JSON-LD format are immutable and cannot be modified after they have been published. They can only be replaced by a new Self-Description (deprecated) or given another non-active status.

A Self-Description with wrong or even fraudulent information can be revoked by the original issuer or a trusted party. Who is allowed to revoke is at the digression of the operators of the different Catalogues. Future rules for Catalogue operators by the GAIA-X organization may apply.

Consequences

Having decided on the possible states reveals new questions that need to be answered in the context of the GAIA-X Catalogue.

The specification of the Catalogue has to describe in detail how the information of the Self-Description state is exposed in its API and search query interfaces.

The state information needs to be distributed between Catalogue instances. There might be disagreements / information gaps between Catalogue instances that need to be resolved.

ADR References

```
--------------

- ADR-001: JSON-LD as the Exchange Format for Self-Descriptions

External References
-------------------

- [LetsEncrypt] https://letsencrypt.org
```

Appendix E: Overview GXFS Work Packages

The project “Gaia-X Federation Services” (GXFS) is an initiative funded by the German Federal Ministry of Economic Affairs and Energy (BMWi) to develop the first set of Gaia-X Federation Services, which form the technical basis for the operational implementation of Gaia-X.

The project is structured in five Working Groups, focusing on different functional areas as follows:

Work Package 1 (WP1): Identity & Trust
Identity &Trust covers authentication and authorization, credential management, decentral Identity management as well as the verification of analogue credentials.

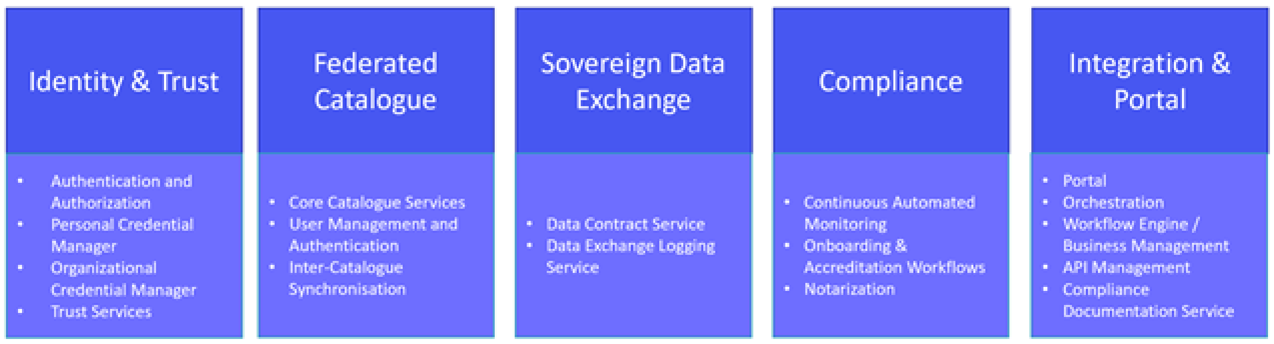
Work Package 2 (WP2): Federated Catalogue
The Federated Catalogue constitutes the central repository for Gaia-X Self-Descriptions to enable the discovery and selection of Providers and their Service Offerings. The Self-Description as expression of properties and Claims of Participants and Assets represents a key element for transparency and trust in Gaia-X.

Work Package 3 (WP3): Sovereign Data Exchange
Data Sovereignty Services enable the sovereign data exchange of Participants by providing a Data Agreement Service and a Data Logging Service to enable the enforcement of Policies. Further, usage constraints for data exchange can be expressed by Provider Policies as part of the Self-Description

Work Package 4 (WP4): Compliance
Compliance includes mechanisms to ensure a Participant’s adherence to the Policy Rules in areas such as security, privacy transparency and interoperability during onboarding and service delivery.

Work Package 5 (WP5): Portal & Integration
Gaia-X Portals and API will support onboarding and Accreditation of Participants, demonstrate service discovery, orchestration, and provisioning of sample services.

All together the deliverables of the first GXFS project phase are specifications for 17 lots, that will be awarded in EU-wide tenders:


Identity & Trust
Authentication and Authorization
Personal Credential Manager
Organizational Credential Manager
Trust Services
Federated Catalogue
Core Catalogue Services
User Management and Authentication
Inter-Catalogue Synchronisation
Sovereign Data Exchange
Data Contract Service
Data Exchange Logging Service
Compliance
Continuous Automated Monitoring
Onboarding & Accreditation Workflows
Notarization
Integration & Portal
Portal
Orchestration
Workflow Engine / Business Management
API Management
Compliance Documentation Service

Further general information on the Federation Services can be found in [1].