23Q4

Digital Registries

Developed by Frank Grozel (UNCTAD), Ingmar Vali(ITU), Tambet Artma (ITU), Saurav Bhattarai (GIZ), Dr. P. S. Ramkumar (ITU), Rauno Kulla (UNCTAD)

1 Version History

The version history table describes the major changes to the specifications between published versions.

Version	Authors	Comment
0.7	Frank Grozel, Ingmar Vali, Tambet Artma, Saurav Bhattarai, Dr. P. S. Ramkumar, Rauno Kulla.	Initial Revision
0.8	Frank Grozel, Ingmar Vali, Tambet Artma, Saurav Bhattarai, Dr. P. S. Ramkumar, Rauno Kulla. Reviewers: Neil Roy, Aare Lapõnin, Amy Darling	Applied feedback from techni review
0.9	Ingmar Vali, Sebastian Leidig, Frank Grozel, Tambet Artma Technical Reviewers: Tony Shannon, Saša Kovačević, Riham Moawad, Riham Fahmi, Aare Laponin, Manish Srivastava, Palab Saha, Surendra Singh Sucharia, Arvind Gupta, Gayatri. P., Shivank Singh Chauhan, Gavin Lyons Reviewers: Steve Conrad, Wes Brown, Valeria Tafoya	Future consideration section analysis and conversion to requirements. Fine tuning, and chapter reorganization.
1.0	Ingmar Vali Reviewers: Steve Conrad, Wes Brown, Valeria Tafoya	Final edits to align content to specification template for GovStack 1.0 release

2 Description

This section provides context for this Building Block.

The Digital Registries Building Block provides services to other Building Blocks and to external systems, to store and manage data/claims on any entity (persons, places, and things) in forms of uniquely identifiable records in a database.

For example, these records could contain health and medical information, ownership of property, vehicles, money, qualifications, birth/expiry of people and entities, land surveys, manufacturing information of vehicles and equipment, banking and commercial transactions, etc. Given the diversity of such information, this Building Block provides services useful to abstract the structure, linkages, and grouping of information into various records and collections such as financial, legal, medical, social, educational, commercial, etc., as needed.

The Building Block provides the capability to capture, store, search, distribute, and present data with zero or minimal need for software development. It also maintains and reports logs of all operations taking place on database schemas and data. It contains various functional components, and data resources to abstract away all the details and complexity, and to expose capabilities as service-APIs to external Building Blocks/applications.

The Digital Registries Building Block is an optional Building Block for other GovStack Building Blocks that have the need to store information. Any traditional database platform could be used alone or in combination with Digital Registries Building Block. The Digital Registries Building Block can operate as a standalone service and could be implemented as one centralized instance per domain, containing multiple registries in one instance, or many instances per domain, each database in its own server.

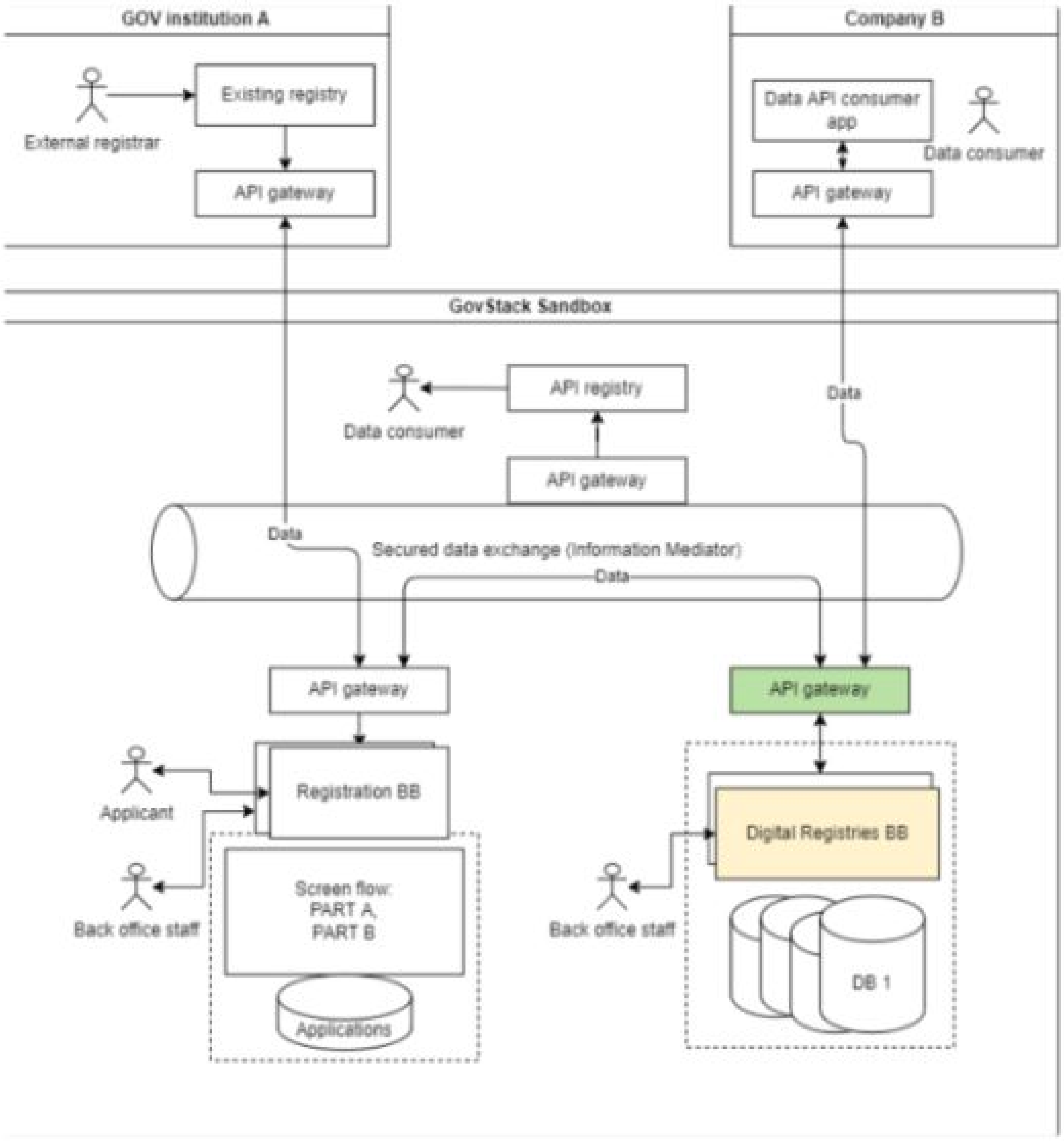


Illustration 1- Digital Registries Building Block in GovStack sandbox

3 Terminology

Terminology used within this specification.

Term	Description
Registry	A paper-based or electronic database (centraliz or decentralized, i.e. blockchain) where claims stored and can be consulted.
Registration	Process through which an entity gets claims recorded in a registry.
Entity	A thing with distinct and independent existence, such as a person, organization, or device.
Claim	An attribute asserted by an entity, about itself or another entity.
Asserter	An entity that asserts a claim.
Registrar	An entity that is authorized to register, in a regis claims submitted by an applicant.
Applicant	Entity that requests the registration of claims in registry.
Operator	A registrar or a staff of a registrar who is process the request of an applicant.
Administrator/Analyst	A registrar or a staff of a registrar who is building new registry.
Rules engine	A tool transforming business rules relating to a registry, defined by a human analyst, into machi readable statements.
Automation	A database-level data movement.
Trigger	A record-level automation.

4 Key Digital Functionalities

Key Digital Functionalities describe the core (required) functions that this Building Block must be able to perform.

The Digital Registries Building Block is an application meant to offer fast and intuitive database management functionalities to entities without the need for database experts. The Digital Registries Building Block is simple to use like online Excel with advanced data management and connectivity options for advanced users. Digital Registries Building Block is a multi-tenant platform where users can create and manage new registry databases. Each registry database created in the system will have automatically REST services generated.

The Digital Registries Building Block no-code development platform uses graphical wizards to create and build software, unlike the traditional approach which uses computer programming languages. It is simple to use, similar to online Excel with advanced data management, log, and connectivity options for advanced users. Each register created in the system has a simple User Interface to see and edit data and an API connector with automatically created Open API services for machine-to-machine communication. The Digital Registry System does not contain data capturing and workflow functionality, however, if a user interface and data processing is needed then Digital Registries can be combined with other GovStack building blocks (e.g. the Registration Building Block) as a plug-and-play.

4.1 Administrative/Analyst Functions

The first user of the Building Block is an **Administrator/Analyst** who is building a new registry. The Analyst is the person who is building the new registry database, changing the existing database configuration, or simply administering the API user authorization. The Administrator/analyst is using a web user interface.

The key functions of the Building Block for Analysts are:

1. Create a new register/database (via API or Web user interface);
2. Create and configure the schema of the register (API or Web user interface);
3. Change schema configuration and publish the new version of the database and API services (API or Web user interface);
4. Enter data to the register (API or Web user interface);
5. View data records in the register (API or Web user interface);
6. Update data in the register (API or Web user interface);
7. Import/export data from/to external files;
8. Import/export registry database schema;
9. Create API services;
10. View statistics (API or Web user interface);
11. Inspect transaction log of registry data operations (API or Web user interface);
12. Manage access to registry data. Authorize users to see and edit registry records or data fields (Attribute-Based Access Control management);
13. Share data with other users via e-mail, or via a unique and secure Uniform Resource Locator (URL) sharing can be field level or record level.

4.2 Applicant Functions

The second main user is an **Applicant** who is consuming registry data via other Building Block (e.g. Registration Building Block) screen flow or via Information Mediator Building Block API services.

The key functions of the Building Block for Applicants are:

1. Search data from the register;
2. Read data from the register;
3. Create data in the register;
4. Update data in the register;
5. Delete data in the register;
6. Validate if given content exists in specified register;
7. Read statistics.

5 Cross-Cutting Requirements

This section will highlight important requirements or describe any additional cross-cutting requirements that apply to this Building Block.

5.1 Requirements

The Cross-cutting requirements described in this section are an extension of the cross-cutting requirements defined in the Architecture specification and Security requirements. This section highlights cross-functional requirements for the Digital Registries Building Block and in addition, describes any deviation to the Architecture Building Block cross-cutting requirements.

5.1.1 Citizen-Centric (RECOMMENDED)

Cancel mandatory requirement: "Right to be forgotten: everything must be deletable". This is not a good practice for government registries.

5.1.2 Open (RECOMMENDED)

Cancel mandatory requirement: "Cloud-native, i.e. Docker and Kubernetes". Digital Registries must have also an on-site installation option.

5.1.3 Robust (RECOMMENDED)

Operates in low-resource environments

Cancel mandatory requirement: "Occasional power". In Digital Registries not possible, thus should be optional. This can be solved with backup power resources (UPS) and a generator that keeps the systems running without interruptions.

Cancel mandatory requirement: "Low-reliability connectivity". Client-server systems are not reliable in this situation, instead additional hand held connection-less data capturing devices should be used and data reentered/uploaded to the servers when connection is restored (not covered in this version scope).

5.1.4 Databases must not include business logic (RECOMMENDED)

Cancel mandatory requirement. "no triggers/stored procedures shall be used". Some stored procedures may be needed for database record ID generation.

5.1.5 Privacy and protection of user data (REQUIRED)

Add mandatory requirement. The following requirement should be added to other Building Blocks' cross-cutting requirements: Each owner of the personal data (e.g. citizen) must be able to see who has looked at their personal data in the registry. All captured personal user data must be marked as "personal data". Users can make requests to see the information/logs of accessing personal information. API must be available for authenticated users to see their own personal data audit logs.

5.2 Standards

The following standards are applicable to data structures in the Digital Registries Building Block:

5.2.1 OpenAPI

OpenAPI Version 3.0.0, 3.0.1, 3.1.0.

6 Functional Requirements

This section lists the technical capabilities of this Building Block.

6.1 Administrative/Analyst Functions

- DRS-1: Analysts have the option to create a new registry database by filling in the following information (REQUIRED):
  1. Name of the database;
  2. A short name;
  3. Schema of the database (see DRS-3).
- DRS-2: Analysts can create multiple databases in one system instance. Databases must be linkable with foreign keys. See the foreign key API description example in Appendix 2. Analysts can configure which databases and which fields are linked. In this document and foreign key function, we consider databases as database tables that can be linked with one another. See the example Illustration. User story: As a user, I can browse database content (Data) in the user interface and when databases are linked, then I can click and move from one database/table to another where the corresponding linked data will open in the user interface. In the Digital Registries Data user interface, it should be possible to open another database by clicking on the record ID in one database and all corresponding records from the other Database will open. It is required to have at least two levels of IDs (database ID and field ID) to link the databases. See the example API in Appendix 2. Example: In one registry database we store information about Mother and Child records. In the second registry database, we store information about payments made for the mother. The system must enable a foreign key link between the payment database to the Mother and child record database. Users can click in the payment database record user interface to the Mother ID field and the system user interface should open the corresponding record in the Mother and Child database. (REQUIRED)
- DRS-3: Analysts have the option to add fields to the database schema. Fields of the database must contain at least the following elements (REQUIRED):
  1. Field name;
  2. Field type, at least with the following types:
     1. Text;
     2. Number;
     3. Boolean;
     4. Date/time;
     5. Date;
     6. Time;
     7. File (pdf, doc, etc.). File extensions/types must be configurable;
     8. List/Array/Edit grid (sub-table/array of values inside a field);
     9. Block container (optional, to group fields visually);
     10. List of Values/Catalog (holding value and key).

  3. Field properties (see more in DRS-33)
- DRS-4: Analysts have the option to publish the database. Publishing will reveal the database to users. (REQUIRED)
  - Publish uses versioning. Each publish creates a new version of the database schema and API services;
  - Old database schemas must be available to the users;

    - Data stored in the old database versions must be usable in old versions and in new versions;
    - Analysts can delete database schema versions. Same version API services must be deleted at the same time.
- DRS-5: Analysts must be able to configure the API services per registry database. (REQUIRED)
    - The system automatically creates API services to:
        - create data;
        - read data;
        - update data;
        - delete data;
        - validate data (if exists);
        - update or create data.
    - Analysts can hide/disable API services;
    - Analysts can delete API services;
    - Analysts can copy API services;
    - Analysts can create custom API services;
    - The system generates the API data structure from the dynamic database structure automatically each time a publish is done.
- DRS-6: Authorization to (REQUIRED)
    1. create and manage databases;
    2. API usage per service, per record, per data field;
    3. access to DATA.

  Analysts have the option to manage user rights of a database and data via API and via a user interface.
    - "Any logged-in user" role must be available;
    - "Anonymous" user role must be available;
    - Attribute Based Access Control (ABAC) logic could be used (API, Schema, data fields, record filter, users);
    - Per user, per group of users option must be available.
        - Group is a set of users in a role.
        - Role is a set of rights.
- DRS-7: The system must log all data processing in the database. (REQUIRED)
    - Schema changes must be logged;
    - Data processing (Create, Read, Update, Delete) must be logged;
    - Logs must be visible and searchable to the Analyst via the User Interface;
    - Every data owner (e.g. physical person) has the option to see who has processed his/her data (PersonalData). The function is a standard function for all registries (DRS-14 API example).
    - Change logs are protected with the highest level of integrity (chaining of logs)
    - Database logs could be logged with an external blockchain for additional security (optional).

- DRS-8: Personal Data usage. (REQUIRED)
  System must automatically store all data read requests and store these in the log table.
  - Covers data read events via User Interface and via APIs.
  - Personal Data logs are stored with PersonalData data tag, storing at least the following information.
    - Log ID;
    - Data record ID;
    - Field ID;
    - PersonalDataID (unique and unchangeable identifier of a person);
    - Reader ID- who read the data;
    - Reader name- name or initial of a person;
    - When- the moment when the Personal Data was read.
  - The Personal Data report is visible only for Analysts to see all data read logs and Data Owners (physical persons) to see their own personal data usage log. Input is PersonalDataID field.
  - PersonalData report is usable as an API service (read)
  - System has API for PersonalData reports. API is per registry(database)
  - System must log Personal Data log read events to the log table.
- DRS-9: Analysts must be able to create views of a database. (OPTIONAL)
  - View is a selection of data from a database;
  - View can be opened as OPEN DATA (anonymous user);
  - View can be created and it can be as a base for an API service (Custom API);
  - View is not for changing or deleting data, only for reading;
  - View rights are managed by the user rights management system.
- DRS-10: The option export database schema to JSON file, (optional: XLS file format). (REQUIRED)
- DRS-11: The option to import database schema from JSON file. (REQUIRED); The option to import

  database schema from XLS file. (OPTIONAL)
- DRS-12: Service usage statistics (OPTIONAL)
  - System must record all API service usage information.
  - System must record all searches made in the Registry User Interface and via APIs.
- DRS-13: An analyst must be able to mark a field as PersonalData log object (This field contains personal data). (OPTIONAL)
- DRS-14: An analyst must be able to mark a field as PersonalDataID. This is the data owner's ID. (OPTIONAL)
- DRS-15: An analyst must be able to mark a field as secret- This field contains secret data (credit card number). E.g. secret data (card data) must be encrypted while at REST.
  Information in transit between the Building Blocks is secured with encryption. Information in Transit is described and governed by Information Mediator Building Block. (REQUIRED)
- DRS-16: Analyst has the option to read database schema in the web User Interface. (REQUIRED)

- DRS-33: Analyst has capabilities to configure database field properties (REQUIRED)
  1. API-related field properties:
     - Validation options: required, unique, max, min.
     - blinded/encrypted (DRS-15, DRS-18);
  2. User Interface related field properties:
     - field mask, format,
     - read-only,
     - personal data,
     - enum list selection;
     - blinded/encrypted (DRS-18);
     - multiple value/array. User can add more values (e.g. multi select from catalog list) to the same field. Multiple values are:
       - array type field;
       - validation options- Required, Unique, max, min.
       - Foreign keys (to link other databases in the same ecosystem). See the example schema in Appendix 2.
     - Triggers to automate field content-related actions:
       - create IDs,
       - merge fields,
       - add prefix,
       - suffix,
       - conditional logic,
       - trigger will be activated if certain condition(s) are true,
       - transform-upper/lower case/ javascript);
       - Triggers are automated when a record is created/changed. A trigger is a record-level automation.

- DRS-34: Analyst has the capability to add an encryption key per database. (REQUIRED)
  - Encryption key is used to encrypt and decrypt data (DRS-17).
  - Encryption key can be used by applications to read encrypted data. Each database has a unique encryption key defined by the analyst.
  - Encryption key is blinded in the User Interface.

  If applications want to read encrypted data via API they must know the encryption key. Data is decrypted in the user interface.
- DRS-35: Analyst has the capabilities to automate data exchange between databases internally and externally via API. (REQUIRED)
  - Automation is triggered automatically after a pre-configured time interval as a loop (finishes when all corresponding records have been processed).
  - Automation processes one record at a time.

- Automation has configurable conditions (business rules in Rules Engine). E.g. IF field A = 123 then true. Conditions can be grouped with AND and OR operators.
- Automation is configured by mapping (input, output) registry data fields to:
    - another database in the same instance.
    - API in an external database.

  Mapping involves:
- query part (input)
- answer part (output)

Mapping can be done from many to one and one to many. Mapping may have a transformation option to convert data to another format. E.g. est->EST;
Expected outcome: Automation can be activated automatically when certain conditions are true and the system sends data to another database or to an external API.

- DRS-36: Analyst may have capabilities to use database schema templates so that the registry creation is faster. (OPTIONAL)
    - Schema templates can be shared in the same instance (internal marketplace).
    - Schema templates can be shared in a marketplace.
    - Schema templates can be imported and exported.
- DRS-17: Analyst has a view to see all data in the registry. (REQUIRED)
  Two main views:
    - Main registry records grid view.
    - Record detail view.
        - See data;
        - See documents(open if image, download if other type);
        - Data log view (changes (create, update, delete). Data before and after).
        - Data read view (information about who has looked at/exported the data). Data and data reader information is stored in the log registry.
- DRS-18: Analyst has a view to edit data in the registry. (REQUIRED) Two main views:
    - Main grid (inline editing).
    - Detail record edit view:
        - Edit data;
        - Remove/add documents (upload).

      All data changes are logged.
    - Analyst has option to delete data in the registry.
        - All data changes are logged.
- DRS-19: Analyst can use additional functions to simplify data searching (REQUIRED)
    - Filtering by search criteria by field content.
    - Full-text data search.
    - Order by each data field.
-

- DRS-20: Import data to the registry. Analyst has the option to import information into the database. Import formats are: JSON, CSV, XLS. (REQUIRED)
- DRS-21: Export data from the registry. Analyst has the option to export selected/filtered data from a registry to CSV/XLS, JSON. (REQUIRED)
- DRS-22: Statistical queries. The system should have the ability to (REQUIRED):
  1. Produce standard statistical reports
     - System must show statistics of all registered items in the registry, with various criteria for filtering. For example:
       - Details of registered people
       - Details of registered services
       - Time series: Change in registration of people/services over time
       - Details of change to data elements (audit logs)
     - Generate customizable reports based on the fields registered in the registry.
  2. Allow the analyst/user to analyze data collected in the system in various ways:
     - (Option) Develop functionality to allow custom dashboards for analysts to analyze data within databases.
     - Provide APIs for extracting data from databases to analyze in external data analytics systems (e.g. Tableau).
- DRS-33: Users can share data with other users. Share data with other users via e-mail, or via a unique and secure URL. Sharing must be at a record level and field level. Data sharing can be turned off in the authorization module. Data can be shared with anonymous users. The data shared with anonymous users is Open Data. (REQUIRED)
- DRS-28: Developer has the option to create a new registry database by sending data via API (REQUIRED). Developer is a user who is using API interface.
  1. Name of the database;
  2. A short name;
  3. Schema of the database (see DRS-3).
- DRS-29: Developer can create multiple registry databases into one system instance. (REQUIRED)
- DRS-30: Developer has the option to publish the database. Publishing will reveal the database to users. (REQUIRED)
- DRS-31: Developer must be able to modify API services per registry database. (REQUIRED)
  - The system generates the API data structure from the dynamic database structure automatically each time a publish is done.
  - The system automatically creates API services to:
    - create data;
    - read data;
    - update data;
    - delete data;
    - validate data (if exists);
    -

update or create data.

- Developer can hide API services;
- Developer can delete API services;
- Developer can copy API services;
- Developer can create custom API services.

- DRS-32: Developer has the option to read database schema via API. Developer has the option to read the list API services available per Database. (REQUIRED)

6.2 Applicant Functions

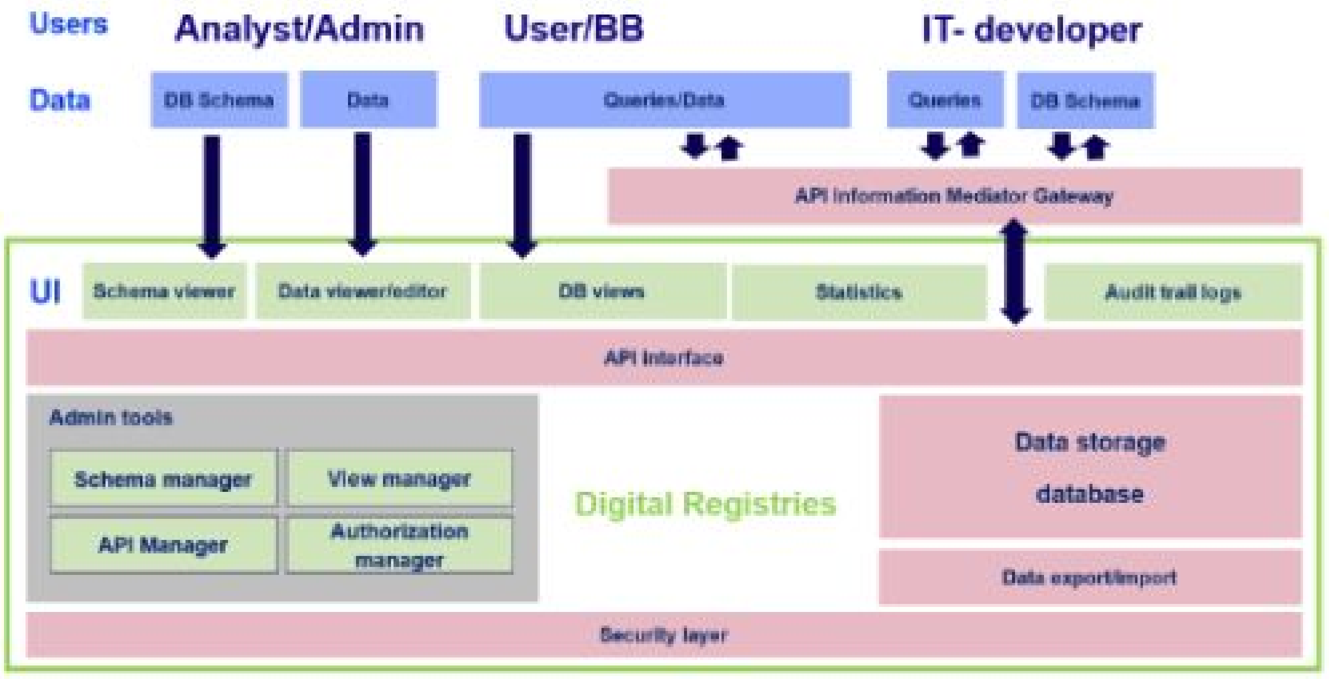
- DRS-23: Building Block must enable client systems to process (CRUD) the database records via Open API services. (REQUIRED)
  - Applicant can search data;
  - Applicant can create data;
  - Applicant can read data;
  - Applicant can update data;
  - Applicant can delete data;
  - Applicant can create or update data.

  Building Block authorizes client systems and users to process data.
- DRS-24: Building Block has the Open API service list (Swagger) to visualize all API services and API service versions. (REQUIRED)
  Client systems must be able to see all API service descriptions including:
  - Description of each field.
  - Example data of each field.

  If possible then the example must be real so that whoever is looking at the API specifications can test the example data in the service (try it).
- DRS-25: System has an API for PersonalData usage report. (REQUIRED)
  - API input must be configurable by the analyst. Input must be a unique identifier of the data owner(e.g. personal identification number).
  - If the registry database schema is designed to store personal data then the analyst must be able to link the personal data to the owner of personal data (e.g. citizen).
- DRS-26: Statistical queries via API. (OPTIONAL)
  - System should make data accessible through the API:
    - Registration Data;
    - Program Data.
  - API should allow querying data with multiple parameters:
    - Date, time ranges;
    - Registered Program.
  - Only authorized data should be available through the API.
- DRS-27: Using viewing event logs- every data owner has the right to see who has looked at their personal data. (REQUIRED)
  - Data owner is a physical person whose personal data is stored in the registry.
  - Data owner has the right to access data reading/processing event logs of the personal data they own. Personal data in a registry is marked accordingly (PersonalData) by the analyst.
  - PersonalData logs are visible via API or via User Interface (PersonalData report).

Building Block Components

The Building Block has a user interface to query and consult the registry data but in most cases, the Applicants are using the end client applications like Registration Building Block to access the registry. Any Building Block can query data from Digital Registries Building Block via APIs if authorization is given.



Digital registries functional components

7 Data Structures

This section provides information on the core data structures/data models that are used by this Building Block.

7.1 Resource Model

The resource model shows the relationship between data objects that are used by this Building Block.

7.2 Data Structures

The Data Structures provide detail for the Resource Model defined above. This section will list the core/required fields for each resource.

7.2.1 Minimum Required Data

Description: The Data Structures can be extended for a particular use case, but they must always contain, at the minimum, the fields defined here.

Fields:

Name	Type	Description	Notes
Database ID	integer	Unique identifier of a database.	Required
Database name	varchar	Name that will define the database content. Name is public.	Required
Schema ID	integer	Database schema ID	Required
Database schema	json object	Database schema. See example in Chapters 7.3.1 and 7.3.2.	Required
Version	numeric	Database version. Each change in schema will produce the next version of the database and API services.	Required
Data ID	integer	Data element unique identifier.	Required
Registry number	varchar	Additional registry identifier. Unique identifier in the registry.	Required
Field type	varchar	Field type: datetime, date, boolean, text, number, file.	Required
Field value	datetime, date, boolean, text, number	Field value, data stored in the field.	Required
Audit log old value	datetime, date, boolean, text, number	Field value before change.	Required
Audit log new value	datetime, date, boolean, text, number	Field value after the change.	Required

8 Service APIs

This section provides a reference for APIs that should be implemented by this Building Block.

The APIs defined here establish a blueprint for how the Building Block will interact with other Building Blocks. Additional APIs may be implemented by the Building Block, but the listed APIs define a minimal set of functionality that should be provided by any implementation of this Building Block.

The GovStack non-functional requirements document provides additional information on how 'adaptors' may be used to translate an existing API to the patterns described here. This section also provides guidance on how candidate products are tested and how GovStack validates a product's API against the API specifications defined here.

The tests for the Digital Registries Building Block can be found in this GitHub repository.

The Digital Registries Building Block may contain multiple registries/databases. The dynamic nature of the database structure requires a standard set of automatically generated APIs for all databases hosted on the platform. The system generates default API method endpoints automatically after each publication of the database schema. A new API service version is generated after each schema publish. Database schema version and API versions are in sync.

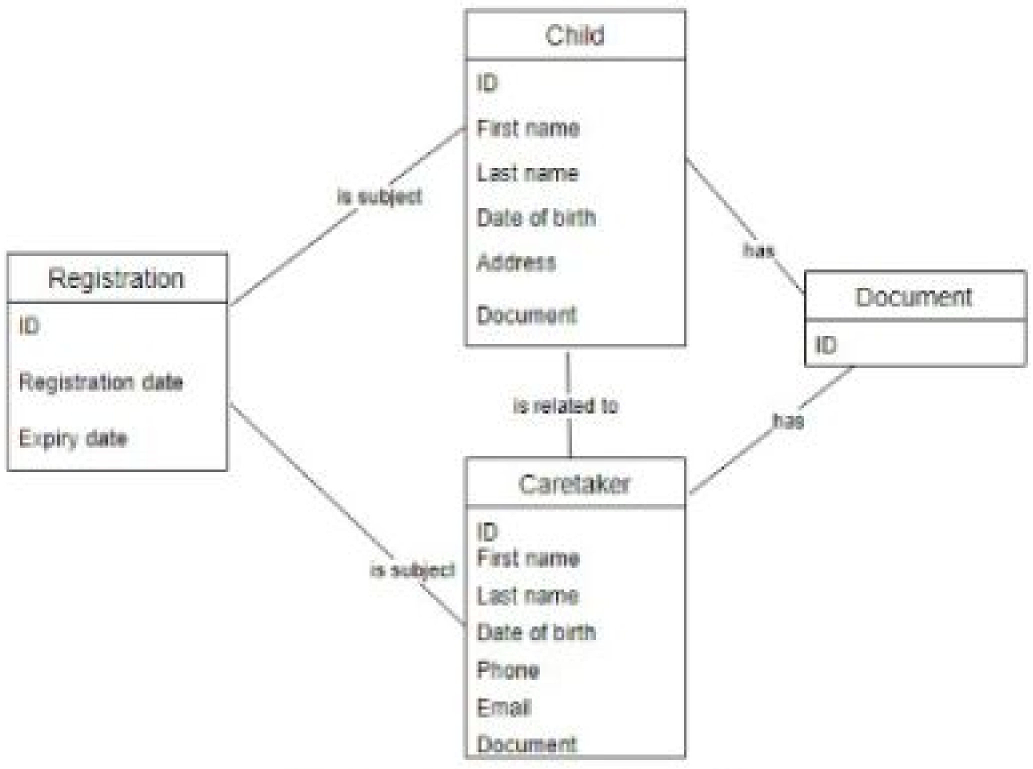
The naming convention and structure of the API endpoint are the following:

/{information type}/{registry acronym or code}/{version}/{API method as a name}.

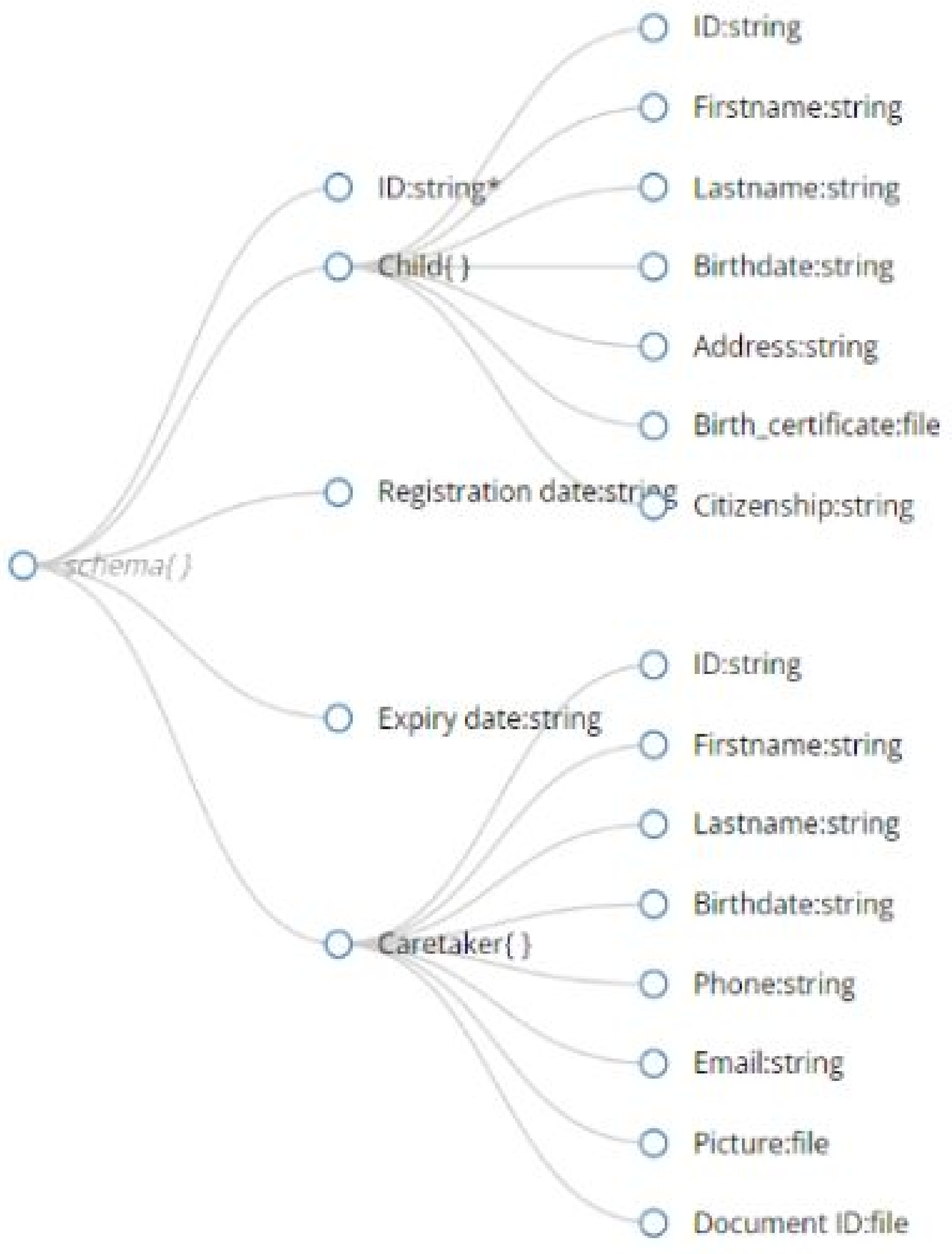
Example 1: /api/data/cr/1.0/create

Example 2: /api/v1/database/modify

Each registry contains a unique set of data and the Building Block enables an Analyst to change the data storage structure/schema on the fly. In the following example API descriptions are generated for one example dataset for the Postpartum Infant Care Program registry, where the Caretaker and infant child are registered and a registration ID is issued.



Example registry database logical data model.

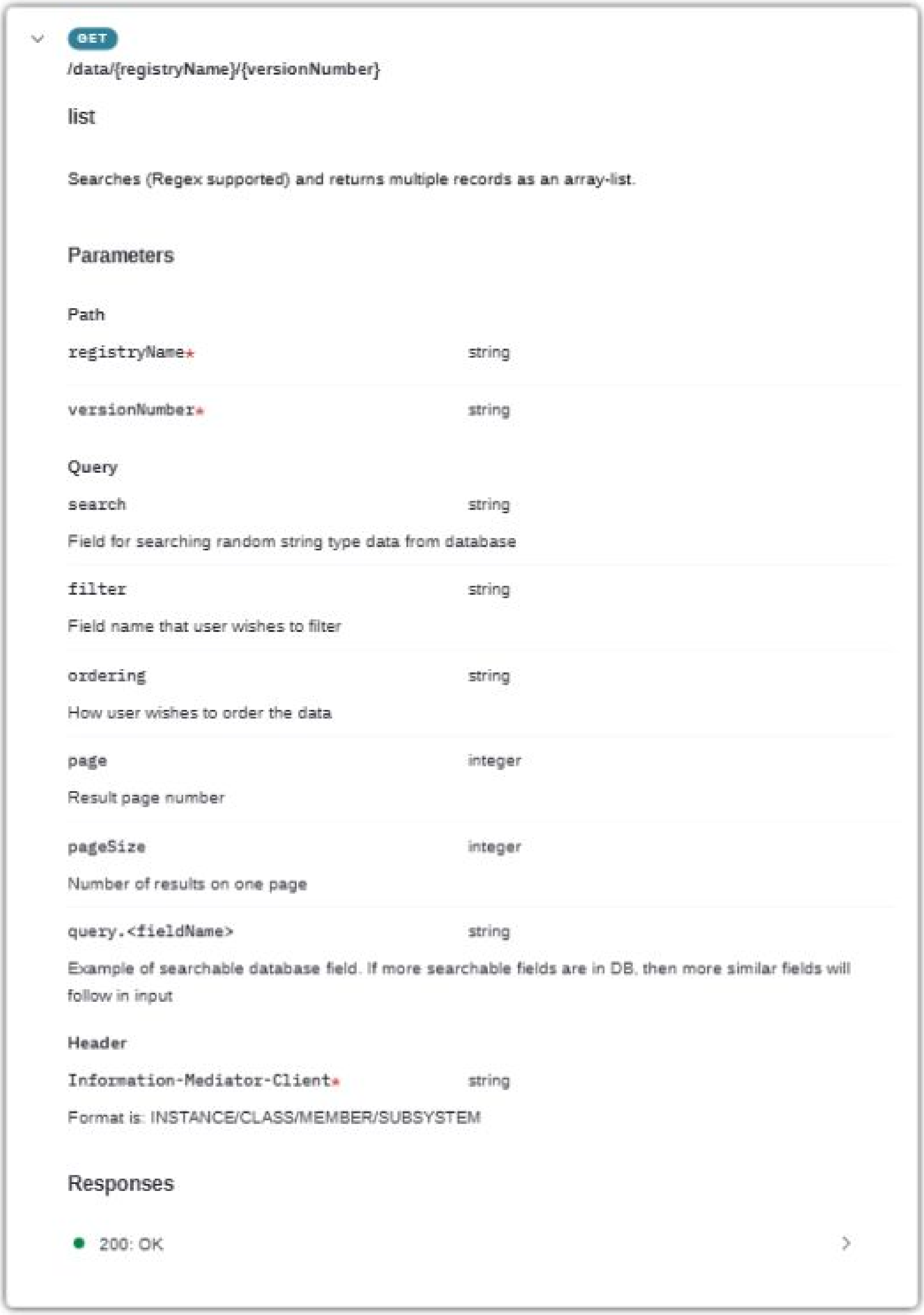


Example registry database Json schema.

Digital Registries Building Block is expected to host the following API services for each database hosted on the platform.

The API is built using a representational state transfer (REST) software architectural style and described in Open API 3 standard using YAML (a human-readable data-serialization language). Request and response body is in JSON (lightweight data-interchange format).

8.1 Administrative/Analyst Functions


GET
/data/{registryName}/{versionNumber}

list

Searches (Regex supported) and returns multiple records as an array-list.

Parameters

Path

`registryName`*	string
`versionNumber`*	string

Query

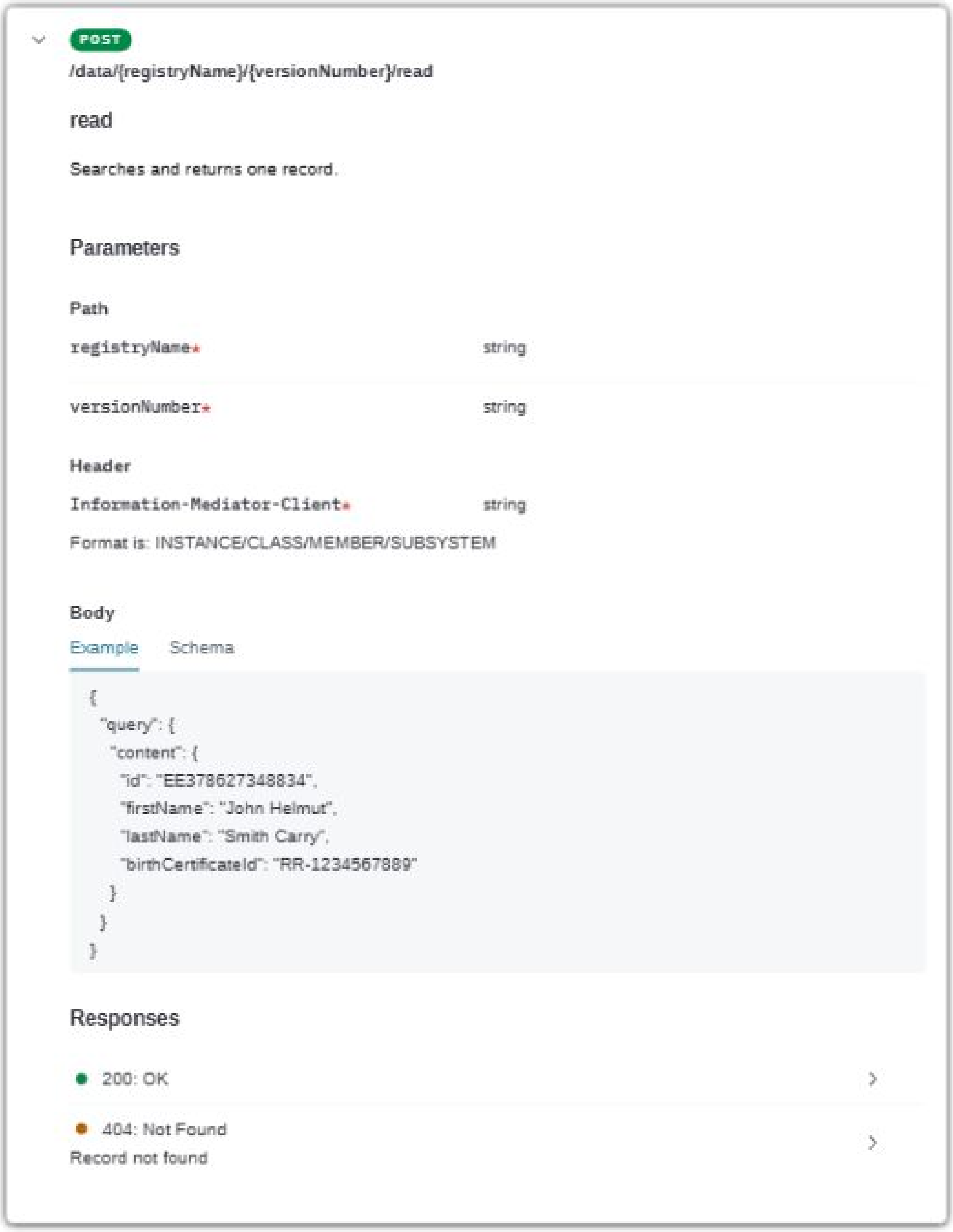
`search`	string
Field for searching random string type data from database	
`filter`	string
Field name that user wishes to filter	
`ordering`	string
How user wishes to order the data	
`page`	integer
Result page number	
`pageSize`	integer
Number of results on one page	
`query.<fieldName>`	string
Example of searchable database field. If more searchable fields are in DB, then more similar fields will follow in input	

Header

`Information-Mediator-Client`*	string
Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM	

Responses

● 200: OK


POST

/data/{registryName}/{versionNumber}/read

read

Searches and returns one record.

Parameters

Path

registryName*	string
versionNumber*	string

Header

Information-Mediator-Client*	string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

Body

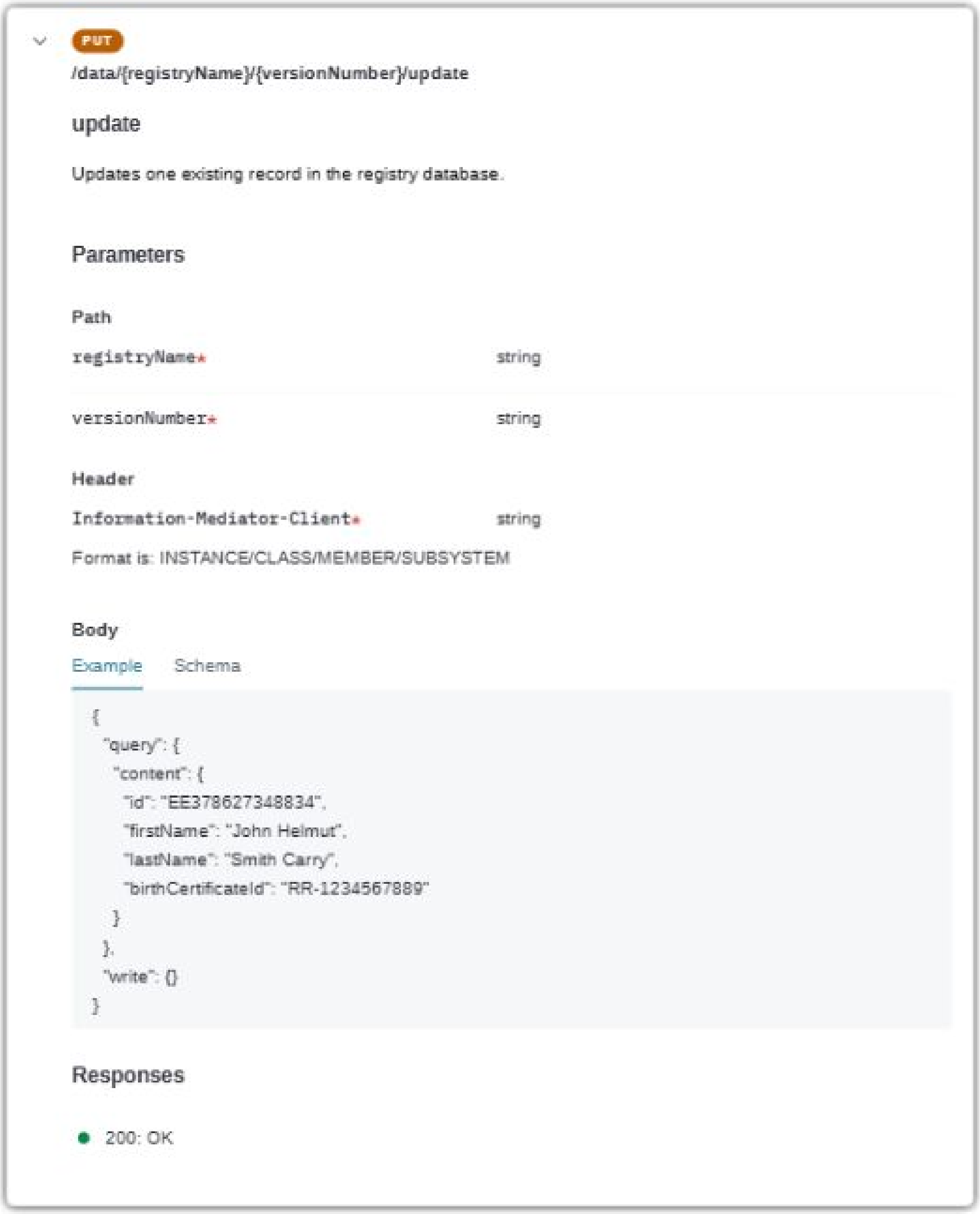
Example Schema

```
{
  "query": {
    "content": {
      "id": "EE378627348834",
      "firstName": "John Helmut",
      "lastName": "Smith Carry",
      "birthCertificateId": "RR-1234567889"
    }
  }
}
```

Responses

● 200: OK

● 404: Not Found
Record not found


PUT

/data/{registryName}/{versionNumber}/update

update

Updates one existing record in the registry database.

Parameters

Path

`registryName`*	string
`versionNumber`*	string

Header

`Information-Mediator-Client`*	string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

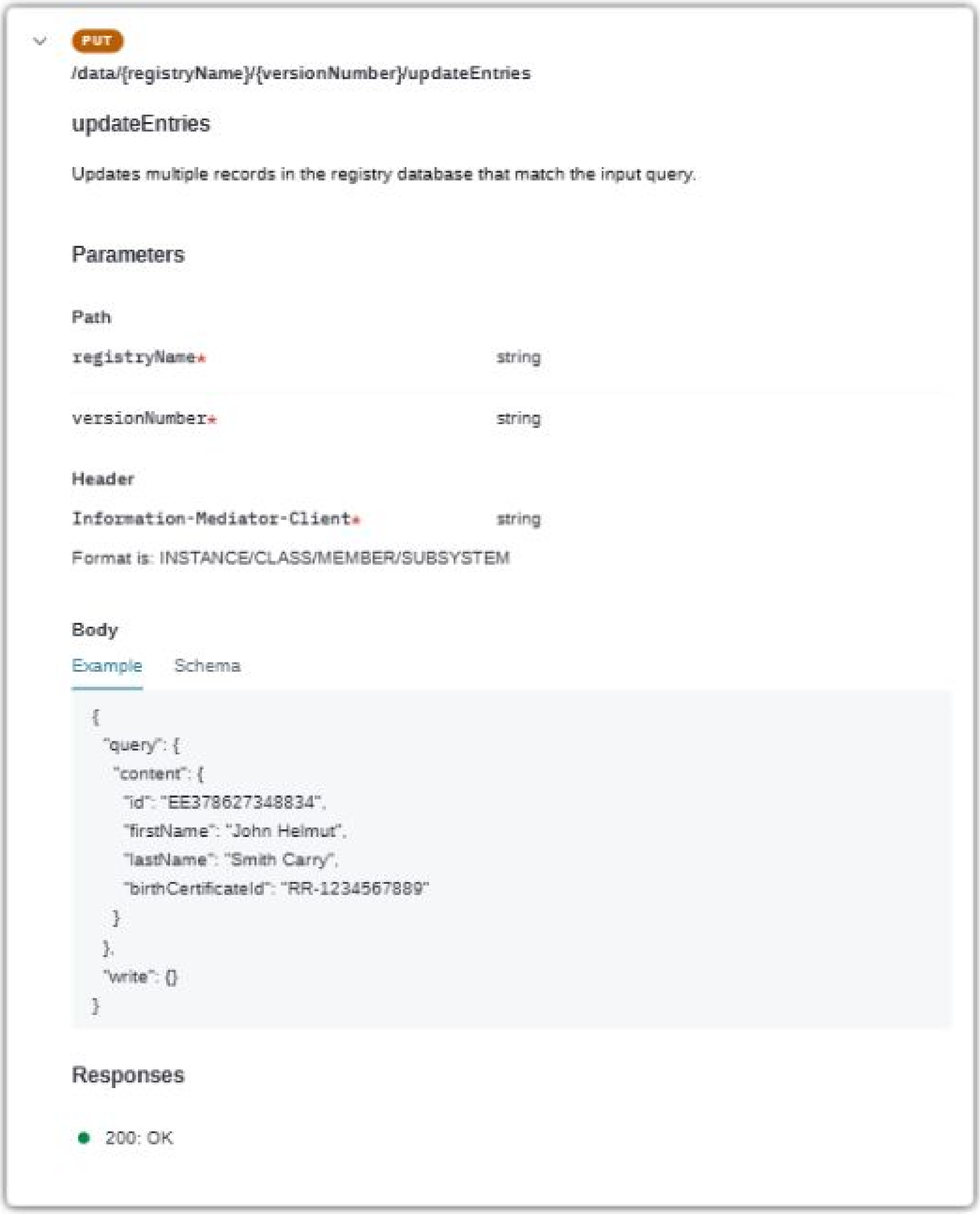
Body

Example Schema

```
{
  "query": {
    "content": {
      "id": "EE378627348834",
      "firstName": "John Helmut",
      "lastName": "Smith Carry",
      "birthCertificateId": "RR-1234567889"
    }
  },
  "write": {}
}
```

Responses

● 200: OK



PUT

/data/{registryName}/{versionNumber}/updateEntries

updateEntries

Updates multiple records in the registry database that match the input query.

Parameters

Path

`registryName`*	string
`versionNumber`*	string

Header

`Information-Mediator-Client`*	string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

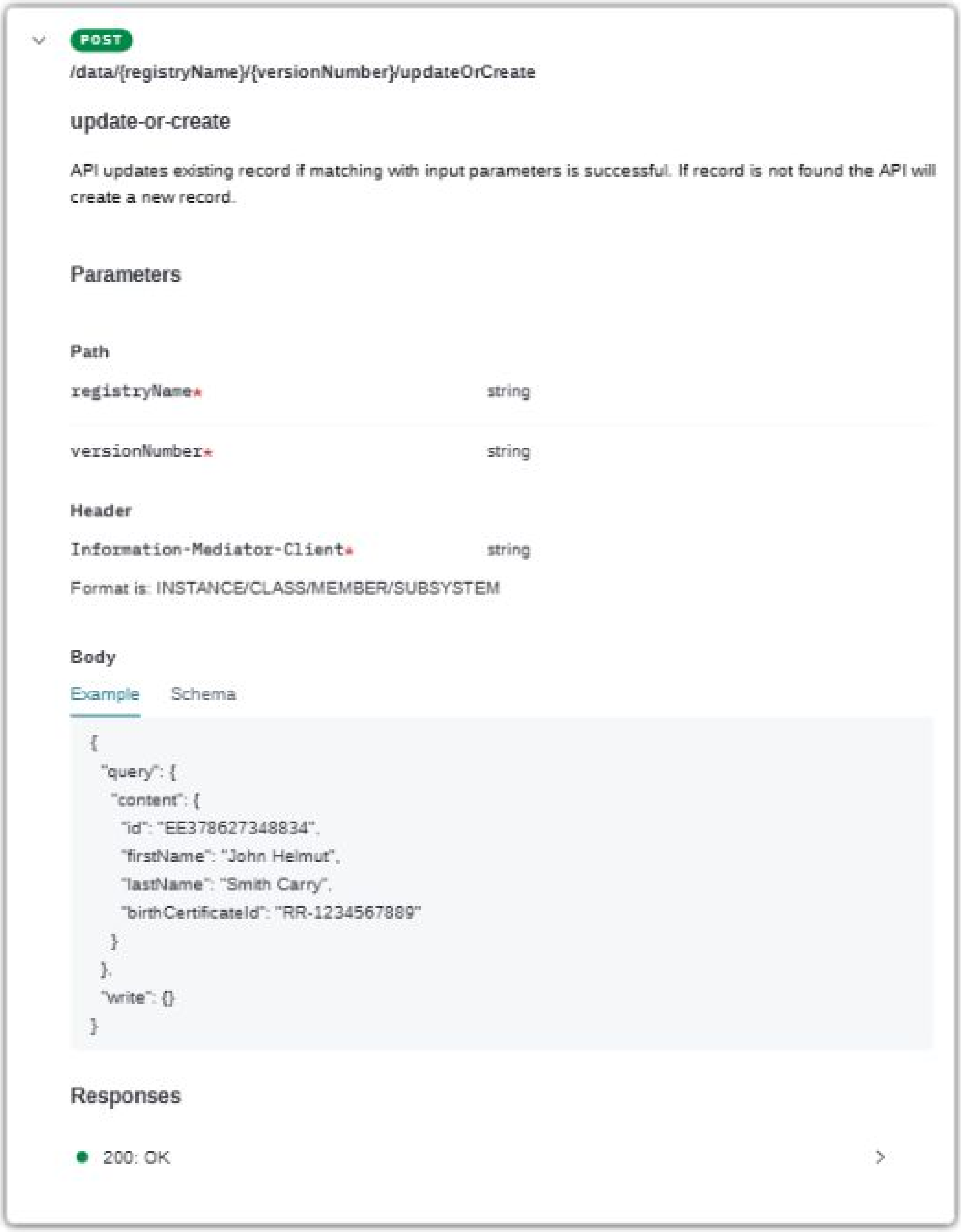
Body

Example Schema

```
{
  "query": {
    "content": {
      "id": "EE378627348834",
      "firstName": "John Helmut",
      "lastName": "Smith Carry",
      "birthCertificateId": "RR-1234567889"
    }
  },
  "write": {}
}
```

Responses

● 200: OK


POST

/data/{registryName}/{versionNumber}/updateOrCreate

update-or-create

API updates existing record if matching with input parameters is successful. If record is not found the API will create a new record.

Parameters

Path

registryName*	string
versionNumber*	string

Header

Information-Mediator-Client*	string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

Body

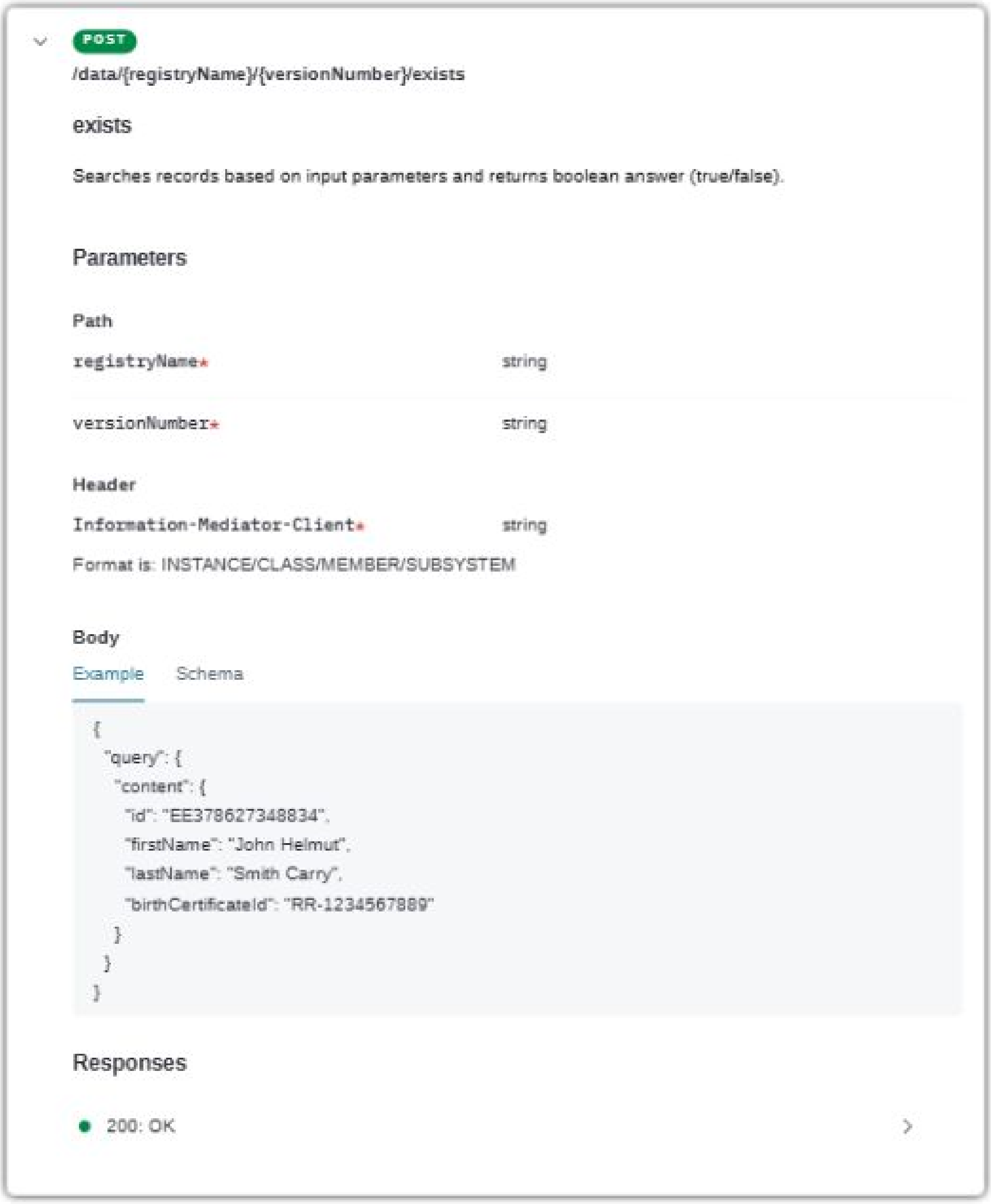
Example Schema

```
{
  "query": {
    "content": {
      "id": "EE378627348834",
      "firstName": "John Helmut",
      "lastName": "Smith Carry",
      "birthCertificateId": "RR-1234567889"
    }
  },
  "write": {}
}
```

Responses

● 200: OK

8.2 Applicant Functions



POST

/data/{registryName}/{versionNumber}/exists

exists

Searches records based on input parameters and returns boolean answer (true/false).

Parameters

Path

registryName*	string
versionNumber*	string

Header

Information-Mediator-Client*	string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

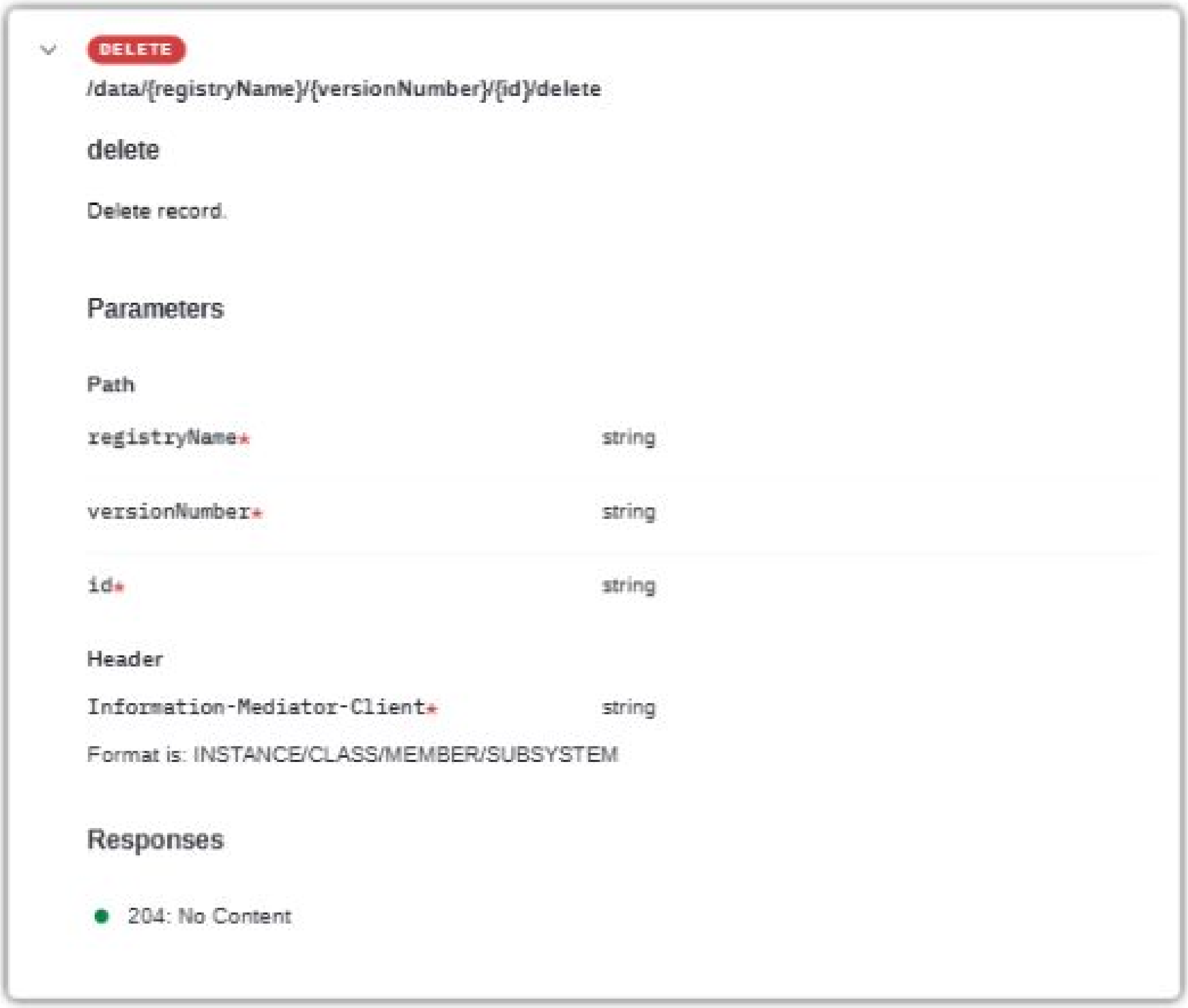
Body

Example Schema

```
{
 "query": {
  "content": {
    "id": "EE378627348834",
    "firstName": "John Helmut",
    "lastName": "Smith Carry",
    "birthCertificateId": "RR-1234567889"
   }
  }
}
```

Responses

● 200: OK



DELETE

/data/{registryName}/{versionNumber}/{id}/delete

delete

Delete record.

Parameters

Path

`registryName`*	string
`versionNumber`*	string
`id`*	string

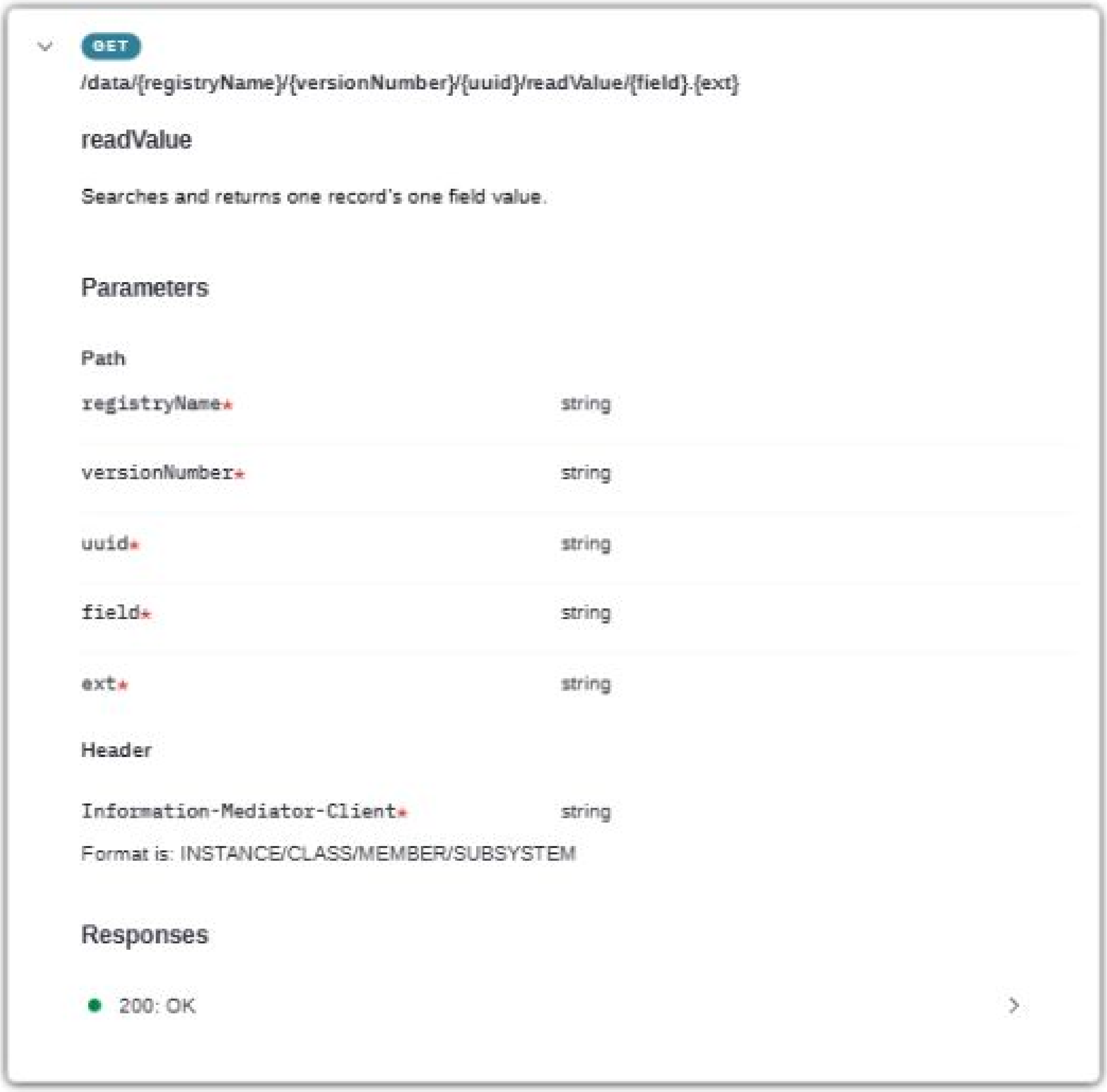
Header

`Information-Mediator-Client`*	string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

Responses

- ● 204: No Content



GET

/data/{registryName}/{versionNumber}/{uuid}/readValue/{field}.{ext}

readValue

Searches and returns one record's one field value.

Parameters

Path

`registryName*`	string
`versionNumber*`	string
`uuid*`	string
`field*`	string
`ext*`	string

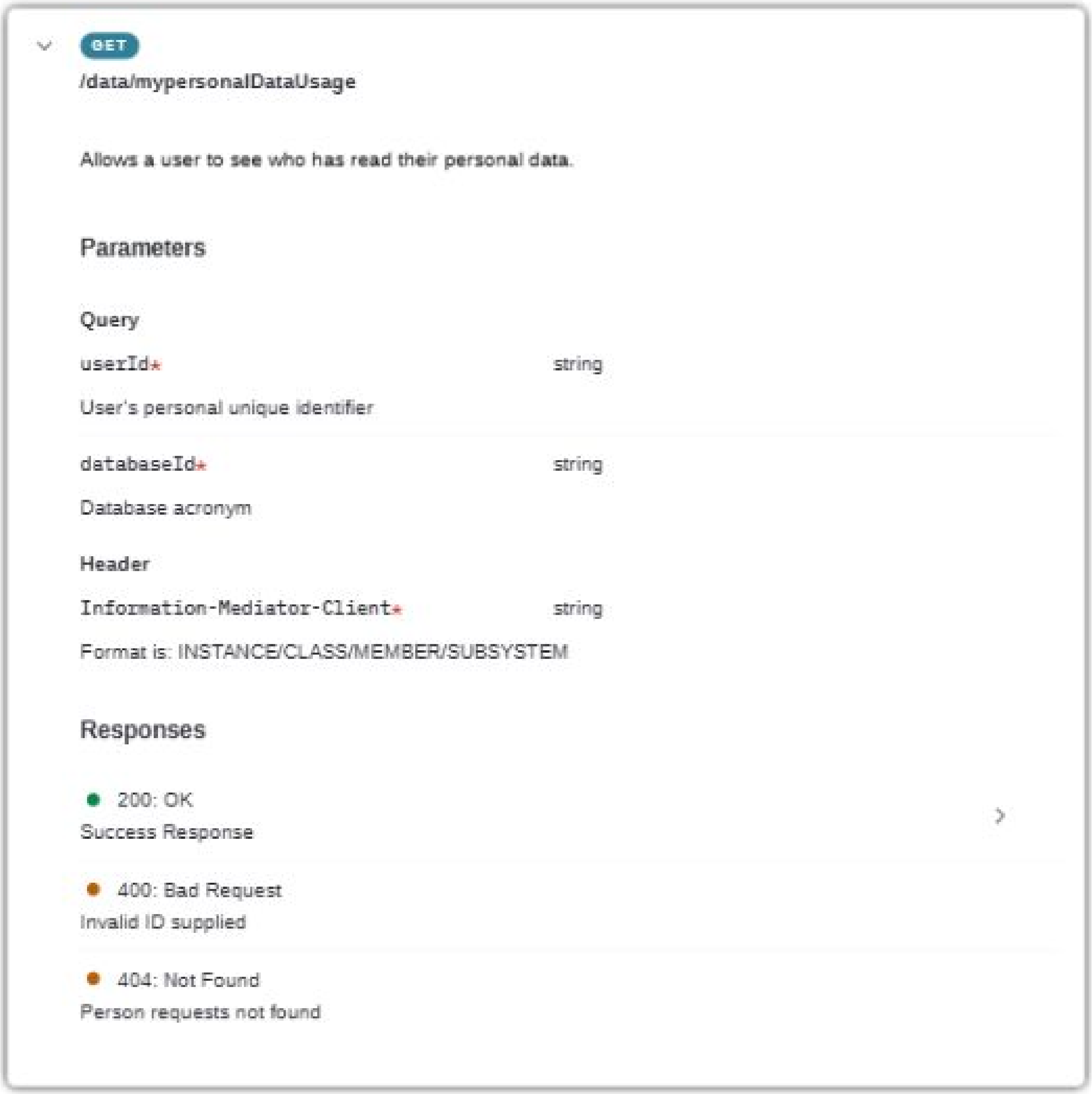
Header

`Information-Mediator-Client*`	string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

Responses

● 200: OK



GET

/data/mypersonalDataUsage

Allows a user to see who has read their personal data.

Parameters

Query

`userId`* string

User's personal unique identifier

`databaseId`* string

Database acronym

Header

`Information-Mediator-Client`* string

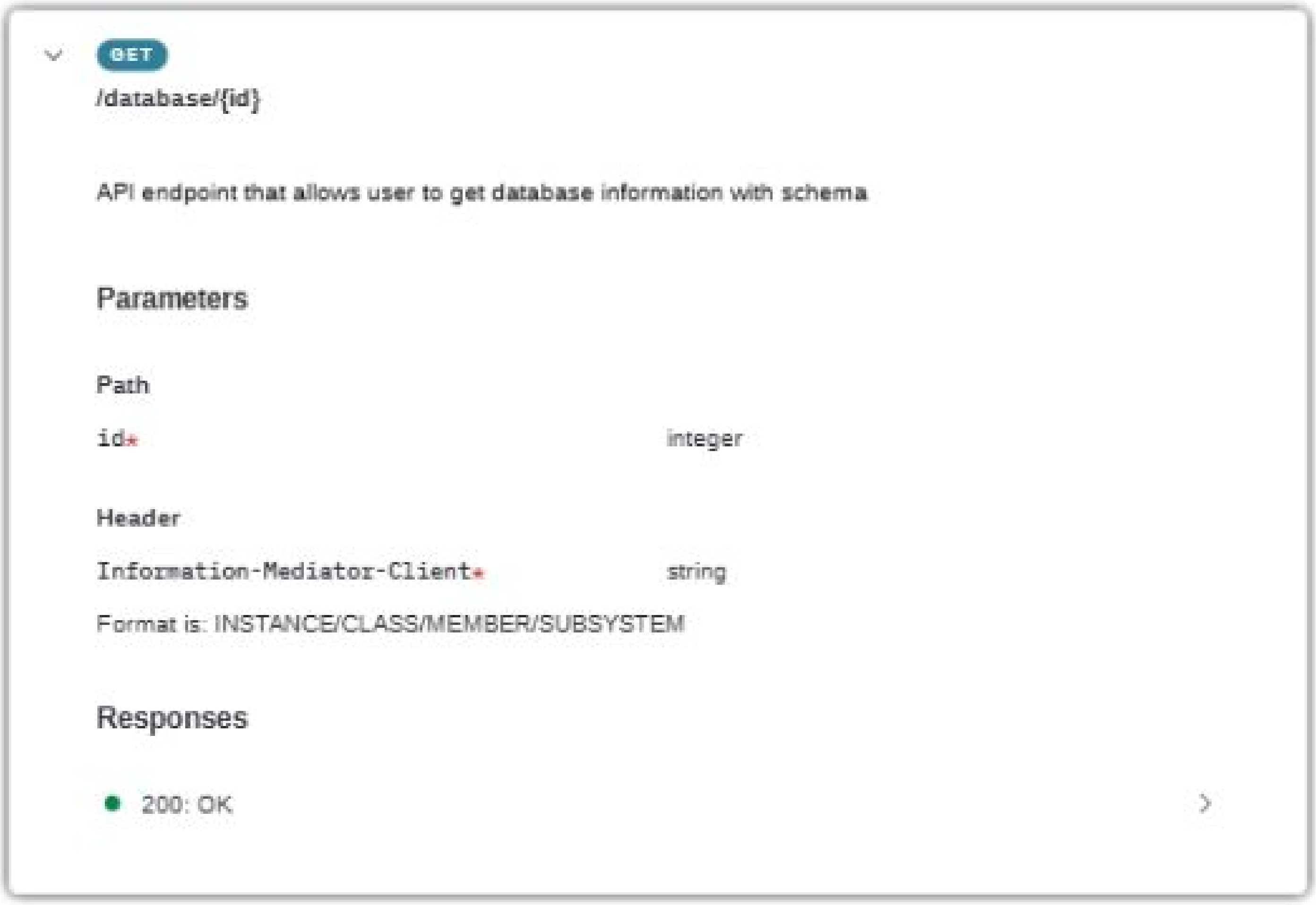
Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

Responses

● 200: OK
Success Response

● 400: Bad Request
Invalid ID supplied

● 404: Not Found
Person requests not found



GET

/database/{id}

API endpoint that allows user to get database information with schema

Parameters

Path

`id*` integer

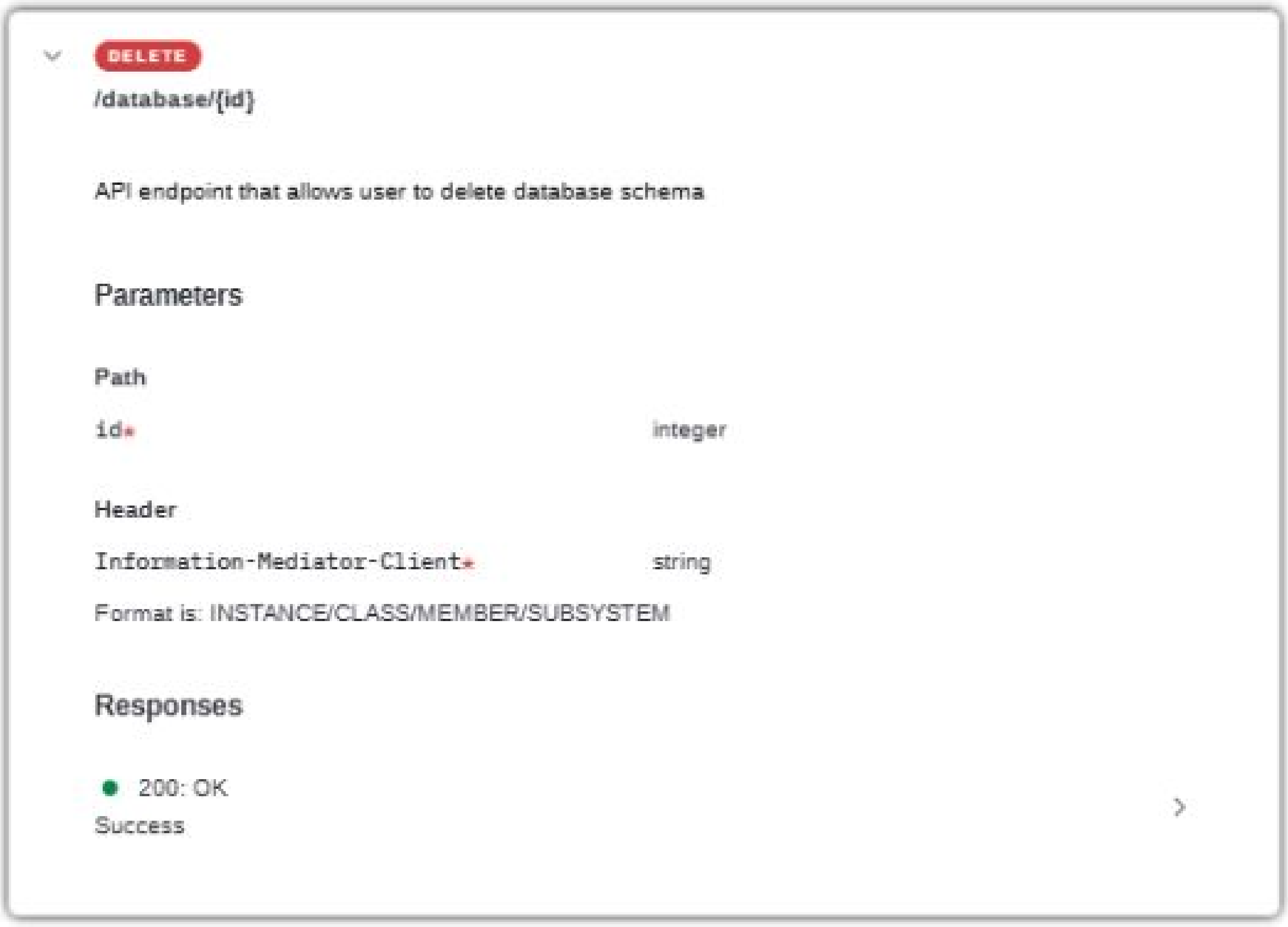
Header

`Information-Mediator-Client*` string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

Responses

● 200: OK



DELETE

/database/{id}

API endpoint that allows user to delete database schema

Parameters

Path

`id*` integer

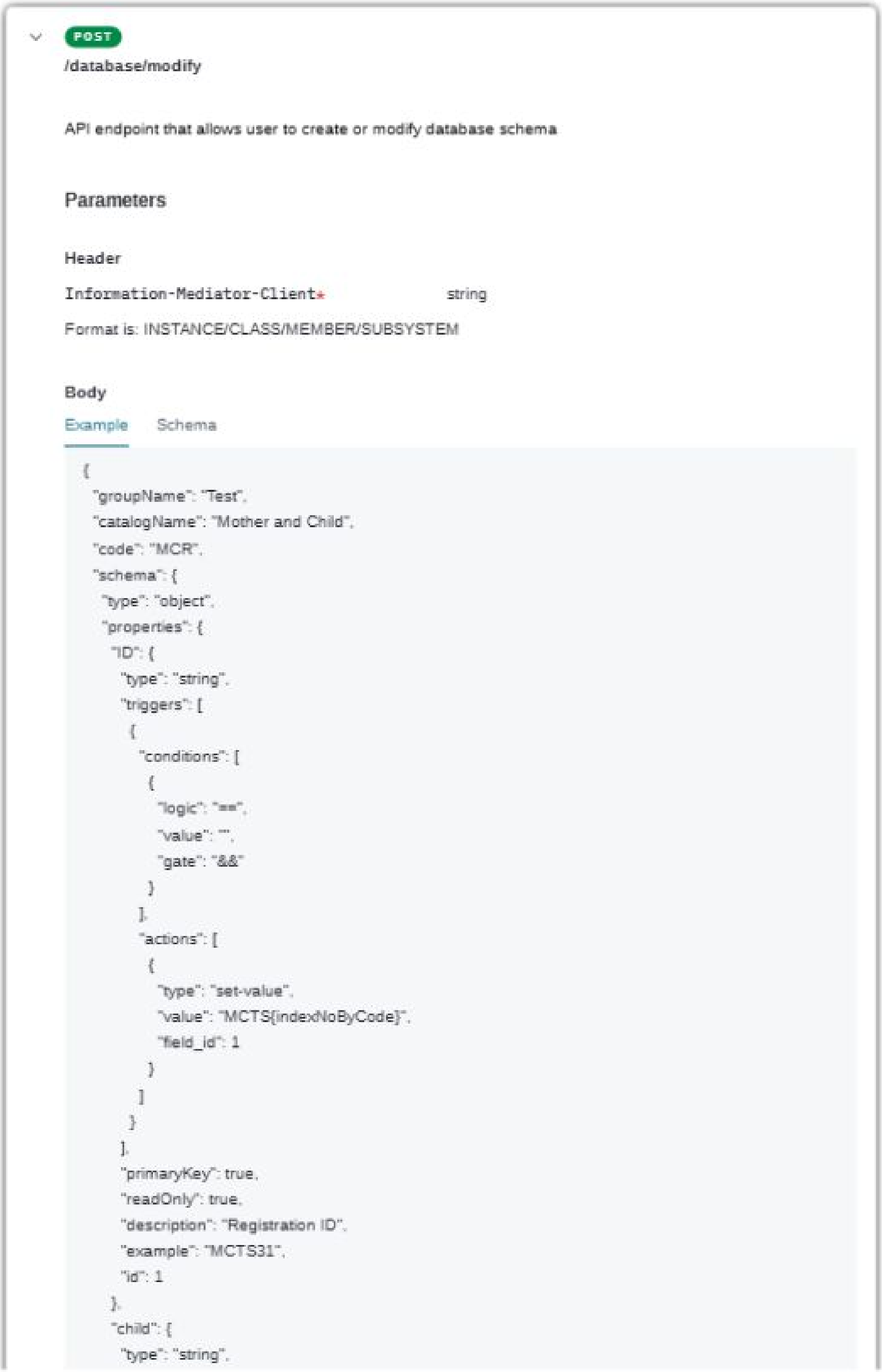
Header

`Information-Mediator-Client*` string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

Responses

● 200: OK
Success



POST

/database/modify

API endpoint that allows user to create or modify database schema

Parameters

Header

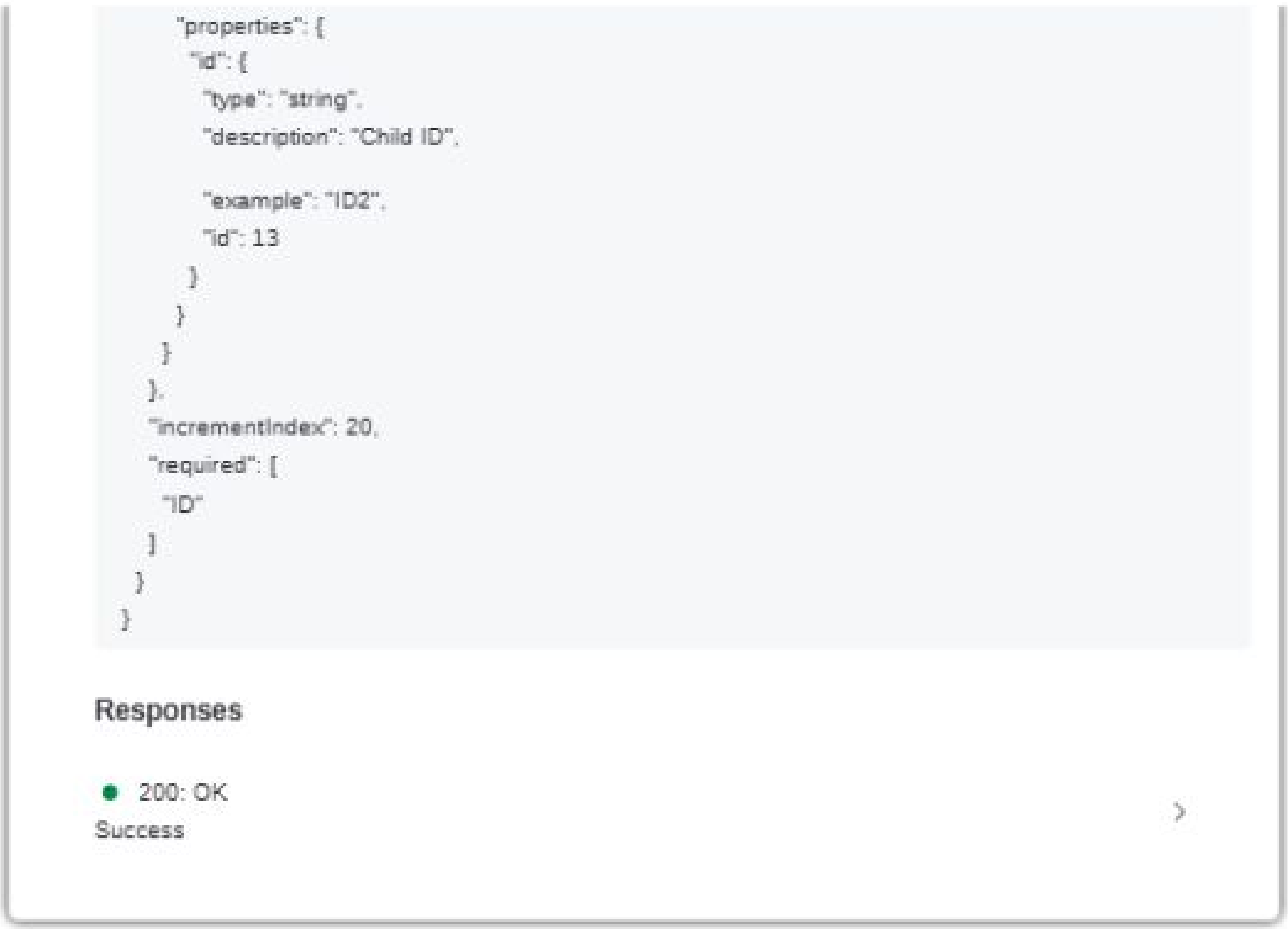
`Information-Mediator-Client`* string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

Body

Example Schema

```
{
  "groupName": "Test",
  "catalogName": "Mother and Child",
  "code": "MCR",
  "schema": {
    "type": "object",
    "properties": {
      "ID": {
        "type": "string",
        "triggers": [
          {
            "conditions": [
              {
                "logic": "==",
                "value": "",
                "gate": "&&"
              }
            ],
            "actions": [
              {
                "type": "set-value",
                "value": "MCTS{indexNoByCode}",
                "field_id": 1
              }
            ]
          }
        ],
        "primaryKey": true,
        "readOnly": true,
        "description": "Registration ID",
        "example": "MCTS31",
        "id": 1
      },
      "child": {
        "type": "string",
```

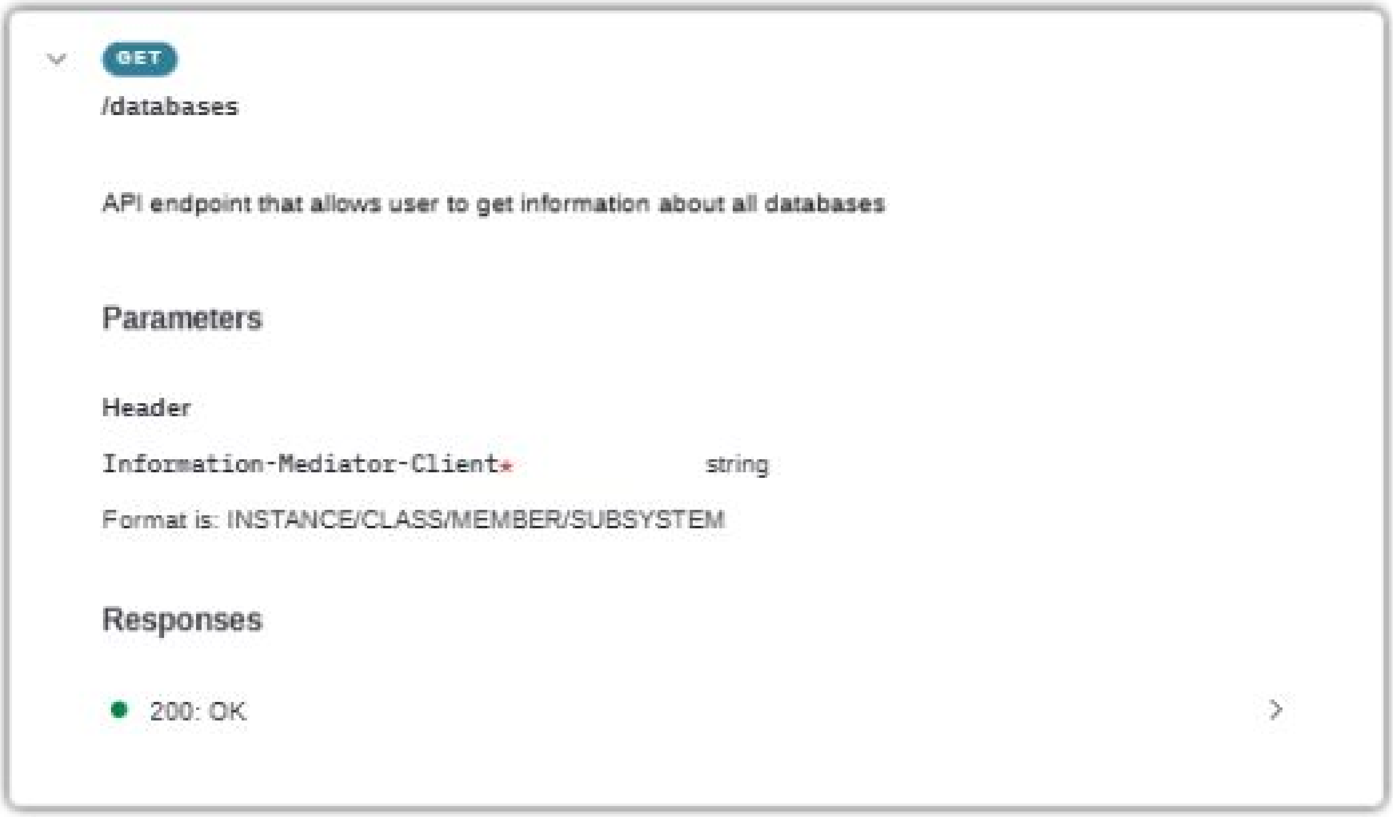


```
        "properties": {
          "id": {
            "type": "string",
            "description": "Child ID",

            "example": "ID2",
            "id": 13
          }
        }
      }
    },
    "incrementIndex": 20,
    "required": [
      "ID"
    ]
  }
}
```

Responses

● 200: OK
Success



GET
/databases

API endpoint that allows user to get information about all databases

Parameters

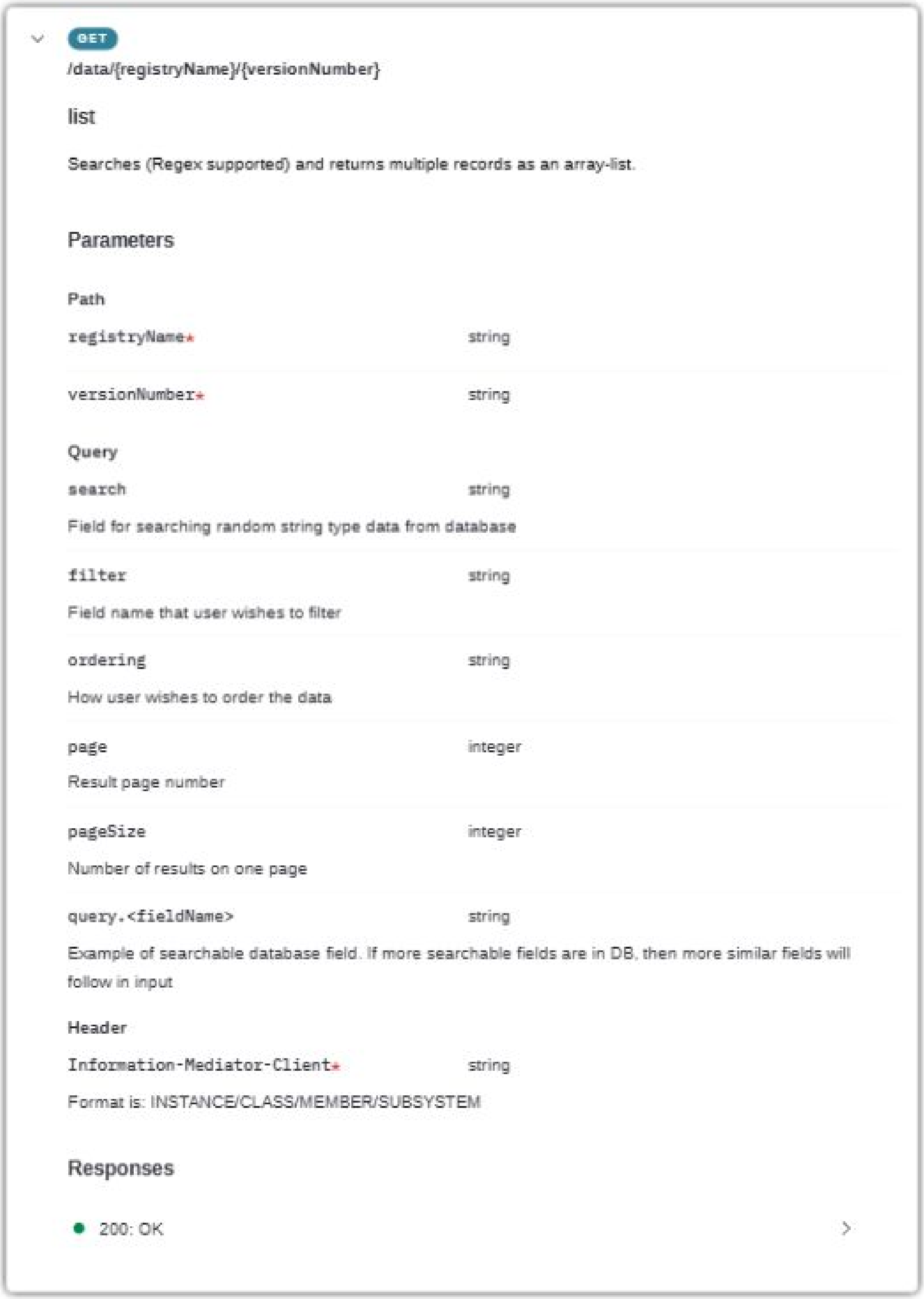
Header

`Information-Mediator-Client`* string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

Responses

● 200: OK


GET
/data/{registryName}/{versionNumber}

list

Searches (Regex supported) and returns multiple records as an array-list.

Parameters

Path

`registryName`*	string
`versionNumber`*	string

Query

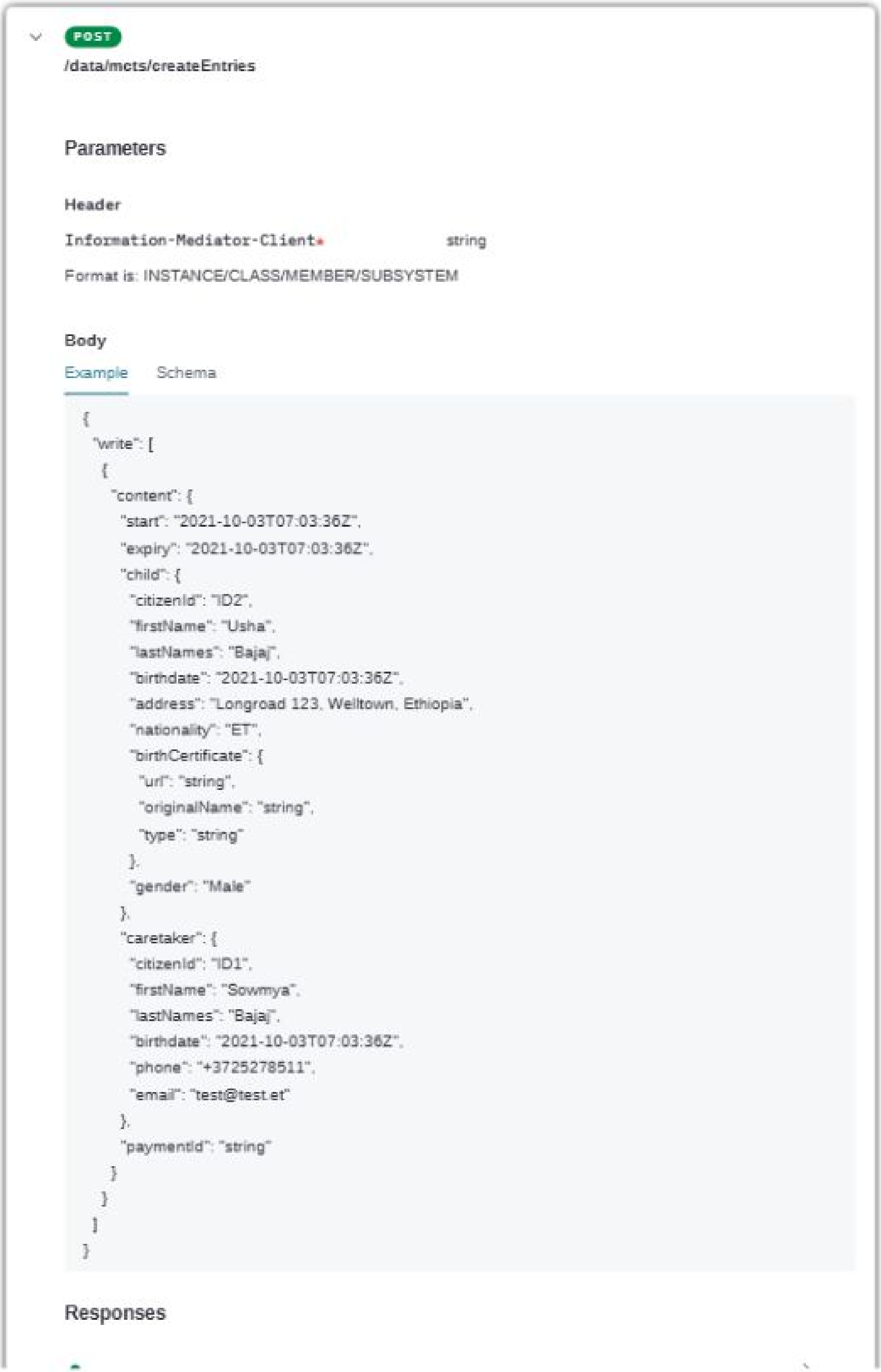
`search`	string
Field for searching random string type data from database	
`filter`	string
Field name that user wishes to filter	
`ordering`	string
How user wishes to order the data	
`page`	integer
Result page number	
`pageSize`	integer
Number of results on one page	
`query.<fieldName>`	string
Example of searchable database field. If more searchable fields are in DB, then more similar fields will follow in input	

Header

`Information-Mediator-Client`*	string
Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM	

Responses

● 200: OK



POST

/data/mcts/createEntries

Parameters

Header

`Information-Mediator-Client`* string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

Body

Example Schema

```
{
  "write": [
    {
      "content": {
        "start": "2021-10-03T07:03:36Z",
        "expiry": "2021-10-03T07:03:36Z",
        "child": {
          "citizenId": "ID2",
          "firstName": "Usha",
          "lastNames": "Bajaj",
          "birthdate": "2021-10-03T07:03:36Z",
          "address": "Longroad 123, Welltown, Ethiopia",
          "nationality": "ET",
          "birthCertificate": {
            "url": "string",
            "originalName": "string",
            "type": "string"
          },
          "gender": "Male"
        },
        "caretaker": {
          "citizenId": "ID1",
          "firstName": "Sowmya",
          "lastNames": "Bajaj",
          "birthdate": "2021-10-03T07:03:36Z",
          "phone": "+3725278511",
          "email": "test@test.et"
        },
        "paymentId": "string"
      }
    }
  ]
}
```

Responses

● 201: Created

POST

/data/{registryName}/{versionNumber}/read

read

Searches and returns one record.

Parameters

Path

`registryName`*	string
`versionNumber`*	string

Header

`Information-Mediator-Client`*	string

Format is: INSTANCE/CLASS/MEMBER/SUBSYSTEM

Body

Example Schema

```
{
  "query": {
    "content": {
      "id": "EE378627348834",
      "firstName": "John Helmut",
      "lastName": "Smith Carry",
      "birthCertificateId": "RR-123456789"
    }
  }
}
```

Responses

● 200: OK

● 404: Not Found
Record not found

9 Internal Workflows

This section provides a detailed view of how this Building Block will interact with other Building Blocks to support common use cases.

9.1 Administrative/Analyst Functions

The Digital Registries building block facilitates the foloowing main internal workflows:

9.1.1 Create a registry database in User Interface

9.1.2 Process registry data in User Interface

9.1.3 Create registry database in API interface

9.1.1 User Story 1 - Create registry database in user interface

As an Administrator/Analyst I want to use a web user interface to create a register database (example registry use case - social security program) so that I can configure and launch the registry database instantly to be used by internet users and client systems (e.g. Registration Building Block, Information Mediator Building Block) via web interface and API.

Actors: Analyst - An administrator user who is creating/changing the registry database schema. The main actor/user in these requirements is the Analyst.

Preconditions:

1. User is authenticated;
2. User is authorized as an admin;
3. User interface is a web interface;
4. User has internet;
5. System has electricity.

Process:

1. Create a new registry database project.
2. Define the database fields.
3. Publish the database.
4. Validate/configure the API services.
5. Manage user rights to access the database and APIs.

Post conditions:

1. System contains a database that is ready to process new data.
2. System has API services to CRUD (Create, Read, Update, Delete) data (and API to validate if data exist).
3. User can enter data to the registry via web user interface (UI).
4. User can see log information in the UI.
5. User can see statistics in the UI.
6. User can give authorization to use the database and process data.
7. System contains a database that is ready to process new data.
8. System has API services to CRUD data (and API to validate if data exist).
9. User can enter data to the registry via web UI.
10. User can see log information in the UI.
11. User can see statistics in the UI.
12. User can give authorization to use the database and process data.

9.1.2 User Story 2 - Process registry data in User Interface

As an Administrator/Analyst, I want to process (Create, Read, Update, Delete) registry data so that I do not have to know the query language.

Actors

- Analyst: the main actor in these requirements is the Analyst/Administrator.
- Data owner: a physical person whose personal data is stored in the registry.

Preconditions:

1. Analyst is authenticated and authorized to use the Building Block and process data in the database;
2. The user interface is a web interface;
3. User has internet;
4. System has electricity.

Process:

1. Analyst searches a record via search or filter function;
2. Analyst selects a record;
3. Analyst processes a record;
4. System stores changes to the Change Log database.

Postconditions:

Processing changes by Analyst are done and log for change is created.

9.1.3 User Story 3- Create registry database in API interface

As an IT developer, I want to Create/update/delete registry database schema via API services.

Actors

- IT developer (Developer): Main actor in these requirements is planning to open a new business program and web form to capture applicants' data. Captured data must be registered in the registry. In this use case, a Developer is any user who is using API services to create and manage registries database.

Preconditions:

1. Developer is using API with a client system or a script that is connected to Information Mediator Building Block. Client system is any Building Block that is using API services via Information Mediator;
2. IT Developer (Information Mediator organization) has been given authorization to Create/update/delete database schema via API services.
3. Developer has internet;
4. System has electricity.

Process:

1. Developer uses a client system to edit the registry database in the Building Block. Developer can:
    1. Create database schema;
    2. Read database schema;
    3. Modify database schema;
    4. Delete database schema and all data in it.

Postconditions:

1. When Developer is authorized to use Building Block API then the Digital Registries Building Block allows processing CRUD (Create, Read, Update, Delete) schema of a registry, and all authorized users can;
2. When Developer is not authorized to process/CRUD the database schema, the system allows to process schema of all databases where an anonymous user has been allowed to edit the database schema (simplification for GovStack Sandbox instance);
3. When a user has no authorization, one can not create nor change (CRUD) any schema in the Building Block.

9.2 Applicant Functions

9.2.1 Process data in API interface

9.2.1 User Story 4 - Process data in API interface

As an Applicant, I want to process CRUD (Create, Read, Update, Delete) data in the registry database.

Actors:

- Applicant - The main actor in these requirements is an applicant via the client system. In this use case applicant is any user who is using a client system (Registration Building Block). For example, a Health Care worker is an applicant in this user story; a mother, using the Registration Building Block. An example client system in this document is Registration Building Block.

Preconditions:

1. Applicant is using client system (e.g. Registration Building Block) that is connected to Information Mediator Building Block;
2. Client system has been given authorization to access Registry to process (CRUD) information;
3. Applicant has been given authorization to access Registry to process (CRUD) information;
4. Applicants are registered in the system and able to use authentication. Applicant is Authenticated by client system or Security Building Block (Authentication).
5. Applicant has internet;
6. System has electricity.

Process:

1. Applicant uses a client system to process data in the registry
   - Applicant can create data;
   - Applicant can read data;
   - Applicant can update data;
   - Applicant can delete data;
   - Applicant can create or update data;
   - Applicant can validate data.
2. System logs all processing events in the dedicated audit registry.

Postconditions:

1. When Applicant is authenticated by a client system (e.g. Registration Building Block) the registry allows processing (CRUD) information from the registry. All users who are authenticated can read data.
2. When a user is not authenticated in the system, the system allows processing (CRUD) data from all databases where an anonymous user has been allowed to process data.
3. When a user has no authorization, one can not process (CRUD) any information in the registry.

10 Other Resources

This section links to any external documents that may be relevant, such as standards documents or other descriptions of this Building Block that may be useful.

10.1 Key Decision Log

A historical log of key decisions regarding this Building Block.

10.2 Future Considerations

A list of topics that may be relevant to future versions of this Building Block.

10.3 Out-of-Scope Assumptions

A list of functions out of the scope of this Building Block.

10.4 Schema Examples

Schema Examples from Data Structures for this Building Block.