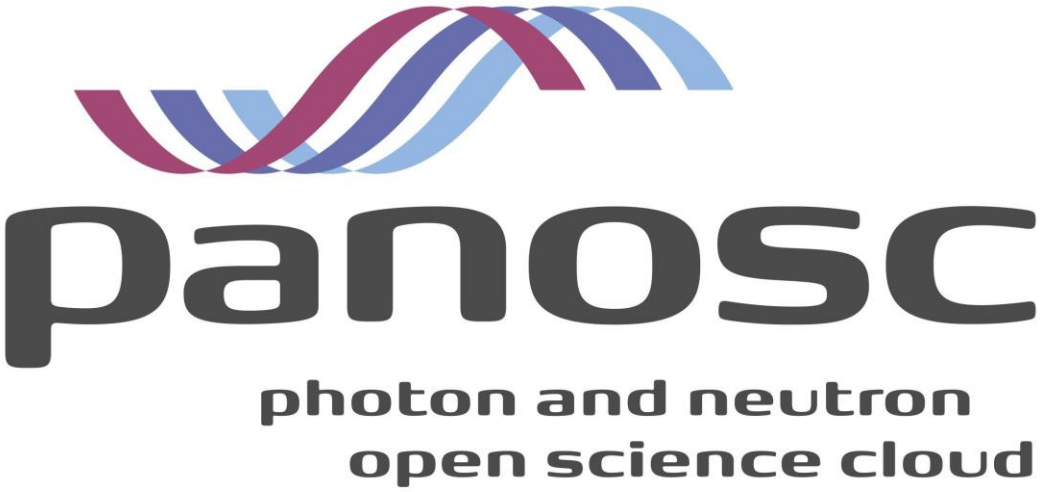
PaNOSC
Photon and Neutron Open Science Cloud
H2020-INFRAEOSC-04-2018
Grant Agreement Number: 823852



Deliverable: D3.3 Catalogue Service (federation of search APIs)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 823852

Project Deliverable Information Sheet

Project Reference No.	823852
Project acronym:	PaNOSC
Project full name:	Photon and Neutron Open Science Cloud
H2020 Call:	INFRAEOSC-04-2018
Project Coordinator	Andy Götz (andy.gotz@esrf.fr)
Coordinating Organization:	ESRF
Project Website:	www.panosc.eu
Deliverable No:	D3.3
Deliverable Type:	Report
Dissemination Level	Public
Contractual Delivery Date:	2022-03-31
Actual Delivery Date:	2022-03-21
EC project Officer:	Flavius Alexandru Pana

Document Control Sheet

Document	Title: Catalogue Service
	Version: 1
	Available at: https://github.com/panosc-eu/panosc
	Files: 1
Authorship	Written by: Tobias Richter (ESS)
	Contributors: Fredrik Bolmsten (ESS), Massimiliano Novelli (ESS), Luis Maia (XFEL), Axel Bocciarelli (ESRF), Alex de Maria (ESRF), Marjolaine Bodin (ESRF), Carlo Minotti (PSI), Emiliano Coghetto (CERIC), Alessandro Olivo (CERIC), Balázs Bagó (ELI-ALPS), Lajos Schrettner (ELI-ALPS)
	Reviewed by: Andy Götz (ESRF)
	Approved: Jordi Bodera (ESRF)

List of Participants

Participant No.	Participant organisation name	Country
1	European Synchrotron Radiation Facility (ESRF)	France
2	Institut Laue-Langevin (ILL)	France
3	European XFEL (XFEL.EU)	Germany
4	European Spallation Source (ESS)	Sweden
5	Extreme Light Infrastructure ERIC (ELI-ERIC)	Belgium
6	Central European Research Infrastructure Consortium (CERIC-ERIC)	Italy
7	EGI Foundation (EGI.eu)	The Netherlands

Table of Contents

Executive Summary

This document marks the delivery of a federated domain specific search as a service for open data from PaN RIs. The search API service is running and serves live data from PaNOSC and ExPaNDS partner sites (ESRF, ESS, ILL and MaxIV). There is even a user web frontend that has been created in collaboration with WP4.

The work builds on the two previous deliverables of PaNOSC WP3. First the API was proposed. In the second iteration the API was improved with input from the proof-of-concept demonstrator and site installations. Deploying the search API as a service for the community as part of this deliverable led to a number of improvements and clarifications in the API, driven by the practical use. In addition, the earlier API definitions left the question of how datasets can be queried using the techniques ontology and how to rank (score) results from different facilities open. This has now been defined and the implementation of the relevant modules at the partner sites has started.

Introduction

This document summarises the development of a federated search service in PaNOSC Work Package 3. The goal of the domain specific search service developed in PaNOSC is to provide a unified way across facilities for scientists in the photon and neutron community to find and filter data using a variety of parameters. ExPaNDS, the INFRAEOSC-05b project, with ten national Photon and Neutron Research Infrastructures (PaN RIs) is working closely with the six PaNOSC partner facilities on common objectives, especially in this data catalogue work package. ExPaNDS RIs have committed themselves to roll out compliant implementations of the search API that can be federated in the same service. This gives users a single point of access to domain specific searches throughout most PaN RIs in Europe.

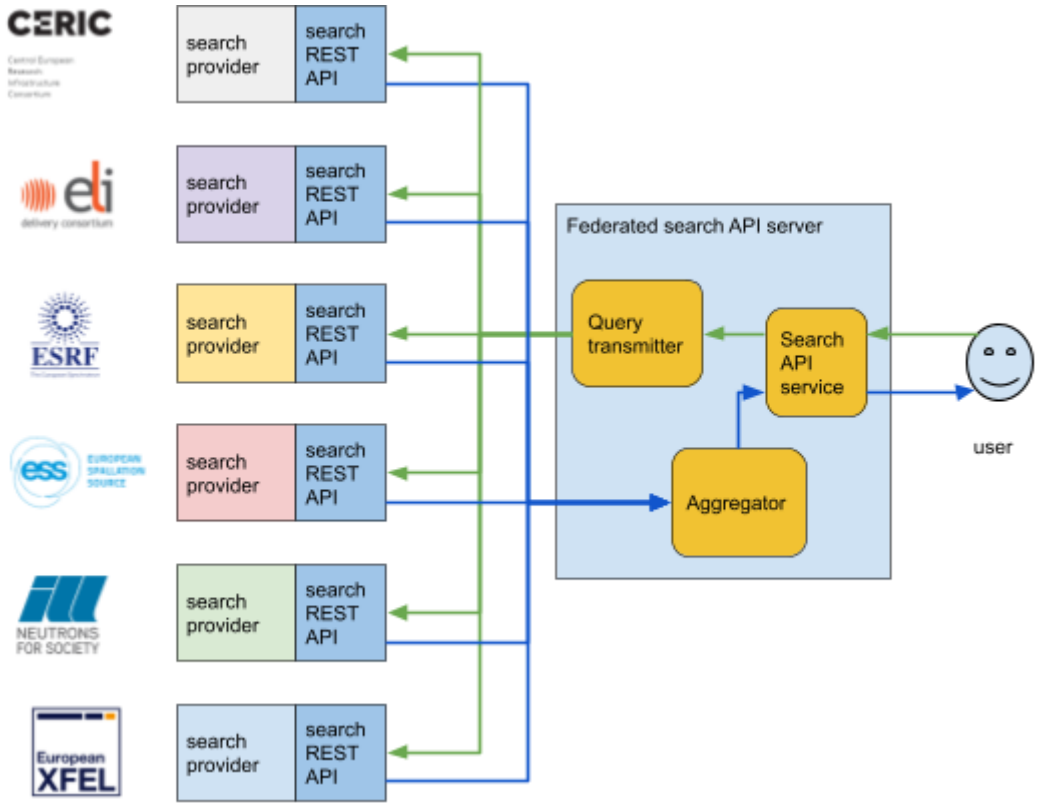


Figure 1: Federated search service architecture.

Following up from the development of the common search API (D3.1) and the development of a demonstrator implementation (D3.2) providing the search as a service is the natural next step. It enables users with a single submitted search query to retrieve matching results from all connected sites in a common result list. After the successful evaluation of the federated search demonstrator this deliverable marks the availability of a production ready service provided towards the EOSC. The activities to achieve this goal can be separated into a refinement of the search API to address any issues that arose in the evaluation phase and work towards the actual implementation. In addition, this deliverable will report briefly on the implementation of software and curation effort at the partner facilities. A more in-depth per-partner review will follow as D3.4.

Third Iteration of PaNOSC Search API

The score field for datasets was introduced in the second iteration of the search API (D3.2). It was set to zero in preparation for the third iteration of the search API which now introduces scoring based on relevance for a given query. The addition of scoring for datasets is crucial as users, using either the local or federated API, expect the most relevant datasets to be listed first. The scoring algorithm was developed to be a standalone service that integrates with search API and can therefore be used by all facilities regardless of underlying infrastructures, ensuring that calculated scores are fair and consistent. Documentation has also been provided to all PaNOSC facilities outlining how scores shall be calculated enabling other partners to implement their own versions.

Additionally, a more advanced search for datasets was developed when it comes to techniques. Previous versions of the search API required users to specify the exact technique stored on a dataset to be able to find it. This is not a trivial task as there exist a wide range of techniques that can be associated with a dataset in essence making it impossible to do wide searches for technique specific datasets. The solution was to develop a structure where more generic techniques attach to more specific ones through a tree structure, this allows users to search for high level techniques and get all associated techniques with it. A standalone service has been created to provide the techniques translation functionality with full integration with the search API, reference and SciCat implementation.

Improvements and documentation updates

The Human Organ Atlas (https://human-organ-atlas.esrf.eu/) is a project run at the ESRF. It enables the wider science community and citizen scientists to explore open imaging data of the human body collected at the ESRF. The project adopted the PaNOSC search API as their chosen backend for querying the image and model repository. The real-world use case helped clarify some earlier decisions around the API. Latest changes implemented in the reference implementation have been driven by the feedback provided by ESRF and their Human Organ Atlas use case.

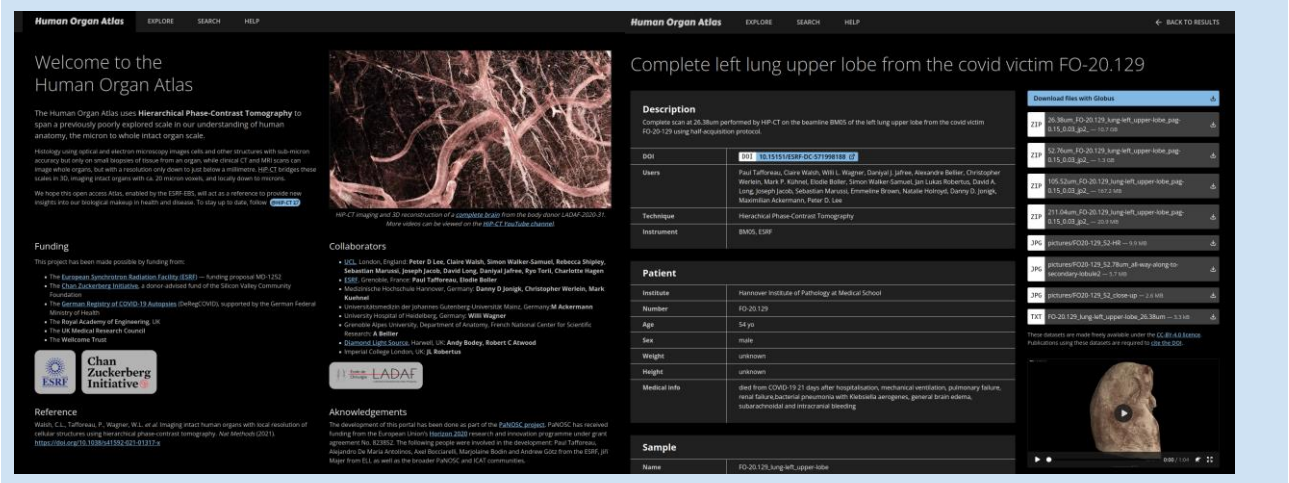


Figure 2: Human Organ Atlas (https://human-organ-atlas.esrf.eu/) browser screenshot

The current search API has been updated to return all the parameters associated in the datasets if the user requests them or specify a condition on any parameter. In prior implementations, datasets were returned only with the parameter specified in the query condition. Now the following query, will return all the parameters of the dataset and not just the one named *"chemical_formula":*

```
{
  "include":[
    {
      "relation":"parameters",
      "scope":{
        "where":{ "name":"chemical_formula" }
      }
    }
  ]
}
```

The following query enables the user to retrieve datasets that specify certain parameters, without imposing any condition on the parameters:

```
{
  "include":[
    {
      "relation":"parameters"
    }
  ]
}
```

In the previous version, this query request would have resulted in a server error.

The documentation has been updated in order to clarify how to retrieve datasets that match one of many conditions, which can be achieved with the following filter syntax:

```
{
  "include":[
    {
      "relation":"parameters",
      "scope":{
        "where":{
          "or" : [
            { "name":"chemical_formula" },
            { "name":"photon_energy" }
          ]
        }
      }
    }
  ]
}
```

Users are now also able to specify multiple conditions on parameters which are applied in logical and. In the previous iteration of the search API, only one condition could be applied. If users would submit a filter with the specified syntax, the service would apply only the last condition and

ignore all the others. For example, if we would like to select all datasets that have two parameters one named chemical_formula and one named photon_energy, the following filter should be submitted:

```
{
  "include":[
   {
      "relation":"parameters",
      "scope":{
        "where":{ "name":"chemical_formula" }
      }
   },
    {
      "relation":"parameters",
      "scope":{
        "where":{ "name":"photon_energy" }
      }
    }
  ]
}
```

This syntax looks somewhat verbose, due to constraints imposed by the underlying technology framework and the data design, but it is a common scheme that some developers will be familiar with.

Differences between "federated" and "local" API

The minimum required set of features for the local Search API are guided by the implementation of the federated search. Each participating facility is required to match this minimum required feature set, so it can be queried by the PaNOSC Federated Search API. In addition, partners can choose to expand the functionality of their implementation to meet local requirements addressing use cases outside the PaNOSC project. This section lists a few identified cases.

Parameters

The federated search does not provide a dedicated end point for providing the complete list of parameter keywords. There is currently a limited list of common parameters that aim to be curated by all partners, published in the last WP3 deliverable. Providing the aggregated list of all potential domain and facility specific parameter keywords would be a long, confusing list of non-homogeneous terms. Here each facility has its own name convention and its own set of parameters. It would not be helpful to refine searches across different facilities. In the long run the aim is to have a more extensive list of parameters curated in a common way, but that requires local effort for the mapping. For the implementation of a local API, for example like in the Human Organ Atlas use case, it may make sense to support such an end point. That does not interfere with the federated functionality.

Pagination

For browsing multiple pages of results, pagination needs to be supported by the API. By design, the federated API should be stateless and simple

in its implementation. This functionality would result in a more complicated code base and demand resources for caching. This led to the decision to exclude pagination from the federated API. Again, at the local endpoint level supporting pagination does not interfere with the federation. So, each facility can decide to implement pagination in their implementation of the PaNOSC search API.

Authentication

The Federated Search API does not require authentication as it serves only public and published datasets. Following the same philosophy, the reference implementation of the Search API does not have any authentication mechanism. Each facility can decide to modify their implementation and add authentication in order to address specific use cases that fall outside the PaNOSC realm. In fact, many facilities have done this and it can provide useful access to embargoed data to the original researchers carrying out the experiments. But exploiting this capability at the federated level would require trust between the user and the federated endpoint and between the federated endpoint and the facilities. For anything more than limited tests this cannot be achieved across organisational boundaries for the foreseeable future.

Techniques

ExPaNDS have been working closely with PaNOSC on the Experimental technique ontology - PaNET (*Collins, Steve P., da Graça Ramos, Silvia, Iyayi, Daniel, Görzig, Heike, González Beltrán, Alejandra, Ashton, Alun, Egli, Stefan, & Minotti, Carlo. (2021). ExPaNDS ontologies v1.0. Zenodo https://doi.org/10.5281/zenodo.4806026*). Especially by defining the terms that should be available in the controlled vocabulary. For the experimental techniques, the idea was to use atomic classes to describe the techniques, here are the four main categories: experimental physical process, experimental probe, functional dependence and technique purpose (naming may change after consultation with all PaNOSC/ExPaNDS partners). The ontology has been defined as a graph structure going from generic terms to more specific ones. Such a structure allows data curators to find the correct term according to the best knowledge about the experiment at that moment in time. Initially an assessment can be made based on basic capabilities of the instrument or facility. A refinement can follow from information about the beamtime proposal. A more experiment specific term can be assigned to a dataset at a later time, highlighting the evolution of the data curation. See figure 3 for a subset of the technique graph to illustrate the relationship between the technique classes.

When a filter on techniques is specified in the PaNOSC search, the API will find and return all datasets that are tagged with the specified technique and all its descendants. A descendant is an ontology term which has the term specified as one of its ancestors according to the ontology graph.

To validate and curate the technique filter, a standalone catalogue independent micro service, has been created under the umbrella of ExPANDS task 3.3.
(https://docs.google.com/document/d/1e9m4KyZOLSkJMO9ex9W_mE_x7nNrzc1JDNz4EfL7U-A section 6.2).

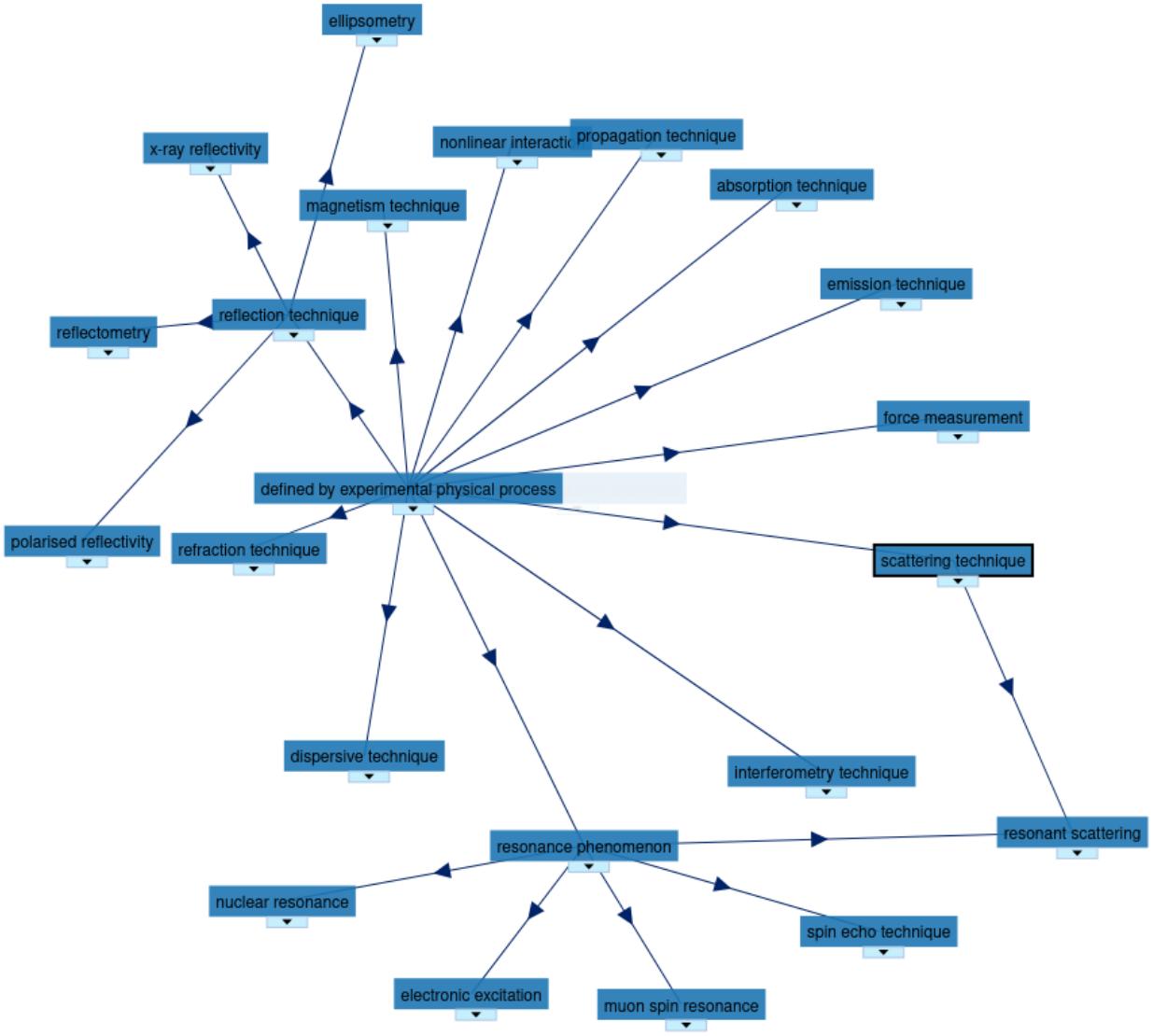


Figure 3: An example section of the ontology tree.

Usage examples

The query API exposes the following REST endpoints:

- ***GET /techniques***
  It returns the techniques cached in memory and pulled from the PaNET ontology. It accepts a Loopback filter (*https://loopback.io/doc/en/lb4/Querying-data.html#filters*) as a query parameter and returns a list of Techniques objects. Following are the fields which belong to the Technique object and that can be used in the Loopback filter:
  - *pid*
    the PURL identifier coming from the PaNET ontology
  - *name*
    the name from the PURL link

- *synonym*
  the list of synonyms for the given pid
- *relatives*
  the list of pids of descendants of the given technique and including it
- *createdAt*
  the time of creation in the memory cache, useful when clearing the cache

- ***GET /techniques/count***
  It returns the number of techniques cached in memory and pulled from the PaNET ontology, following input conditions. It accepts a Loopback filter as a query parameter and returns the numbers of Techniques matching the specified filter. The fields available in the Loopback filter are the same as for the previous endpoint.

- ***GET /techniques/{id}***
  It returns the technique(s) with the provided pid. It also accepts a Loopback filter, with the same available fields as before.

- ***GET /techniques/pan-ontology***
  It accepts a Loopback where condition (*https://loopback.io/doc/en/lb4/Where-filter.html*) and returns another Loopback where condition containing the relatives of the given input condition.

The following example illustrates how the GET /techniques/pan-ontology endpoint works. This endpoint is the only one used in the integration with the search-API. More examples are available in Appendix 2.

Input:

```
{
  “pid” : ”http://purl.org/pan-science/PaNET/PaNET00210”
}
```

REST call:

```
/techniques/pan-ontology?where=%7B%22pid%22%3A%22http%3A%2F%2Fpurl.org%2Fpan-science%2FPaNET%2FPaNET00210%22%7D
```

Output:

```
{
  "pid” : {
    "inq : [
      "http://purl.org/pan-science/PaNET/PaNET00210",
      "http://purl.org/pan-science/PaNET/PaNET01318",
      "http://purl.org/pan-science/PaNET/PaNET01319"
    ]
  }
}
```

Implementation

The implementation has been built on top of the Loopback 4 framework. We reference here the pan-ontologies-API data flow for the PaNET ontology. The data flow in a typical query to the PaNOSC search api containing a condition on techniques is as follows:

1. The search-API receives a Loopback filter per the search-API spec.
2. The where part of the filter, related to, e.g., techniques, is forwarded to the pan-ontologies-API.
3. The pan-ontologies-API fetches the ontology from an external source (if not already cached).
4. The fetched ontology is processed depending on the ontology logic (in the diagram the PaNET one) and the expanded Loopback where condition is returned.
5. The search-API integrates the expanded techniques clause with the original where condition from step two and sends it to the data catalogue. Each individual search API is responsible to translate the full where condition to the syntax used by the catalogue.

In the example in fig. 4, which is specific to the search API SciCat implementation, the search API translate technique pid key to "techniques.pid" as requested from SciCat backend.

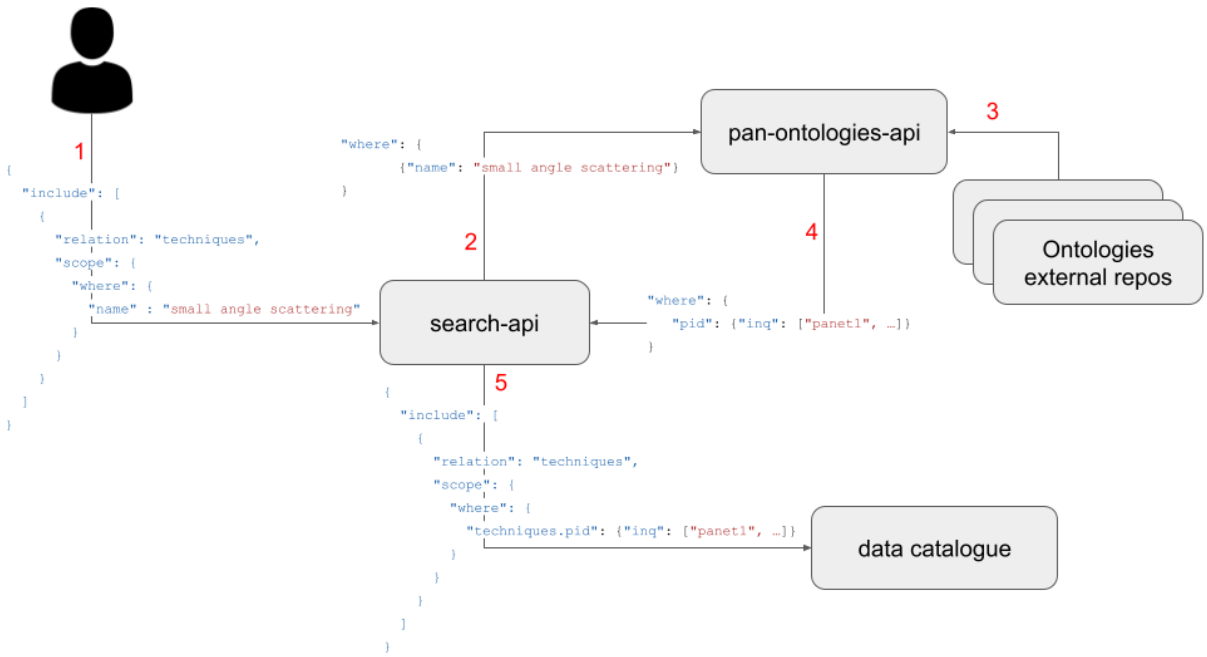


Figure 4: pan-ontologies-API data flow

Given its role in the data model, namely the fact that it does not depend on the data catalogue implementation (e.g. ICAT and SciCat can use the same implementation), the pan-ontologies-API service could work either at the level of the PaN federated search-API service (the middle layer of figure 4), thus becoming a federated service or at the level of the data catalogue of each facility (each box named "search-API" in the bottom layer of fig.3). If the pan-ontologies-API is deployed at the level of each facility, steps 2 and 5 of the data flow aforementioned will need to be implemented individually, while deploying it at the federated level will

enable a transparent integration at every facility, since the response from the pan-ontologies-API can be made compliant with the PaN search-API requirements in the PaN federated search-API service.

A deployment of the PaNOSC search API which includes the techniques integration can be accessed at this URL: https://scicat.ess.eu/panosc-api. Its matching explorer interface is available at https://scicat.ess.eu/panosc-explorer. The techniques microservice used by the deployment mentioned above, is accessible at the URL: https://techniques.scicat.ess.eu/explorer .

Search Results Scoring

Most of the query functionality in the search API provides binary filtering. For example, the search can be limited to datasets that measured a certain chemical, under parameters in a specific range. However, the common free text match that resembles the Google search box, has to rank the results based on "document relevance" in order to meet user expectations. The first version of the search API in D3.1 already specified a metric (expressed in the field named "score") to be delivered from the facility endpoints to allow the service to establish this ranking across the results from different partner sites. With this deliverable we specify how this score metric is to be calculated and used.

Architecture

In contrast to most turn-key software solutions for searching document repositories (like Elasticsearch) we have to find and implement a solution that works well for the task of establishing a relevance score despite the underlying repositories being separate entities in different physical locations under their own administration. The diagram in Figure 5 illustrates the overall infrastructure needed to deliver a PaNOSC federated search with scores. It includes the components required both at the facility level and at the federated level.

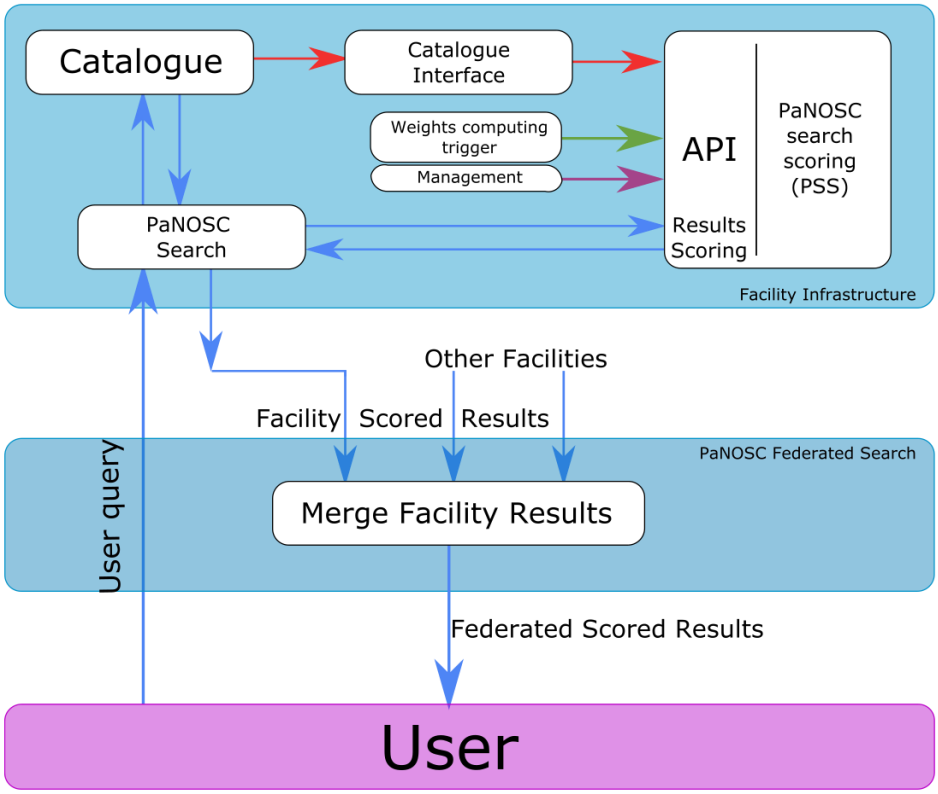


Figure 5: PaNOSC Federated Search Infrastructure with Scoring

The user submits a query together with the number of items (results) requested to the PaNOSC federated search API. The query and the number of items requested are relayed as-is to each connected PaNOSC search API endpoint at all participating facilities.

Each facility retrieves the relevant results from their catalogue based on the query. The results are scored and prioritised locally. Each facility will send the full requested number of scored results back to the federated search layer. The federated search merges together all the results from all the participating facilities, sorts them according to the assigned scores and returns only the most relevant ones to the user. This means that for n connected facilities each one returning r_n results, where $r_n \leq r$, only a result set of r elements is presented to the user and all other results will be discarded. r is the number of results requested by the user. A consequence of the flow highlighted, the number of results the user will be able to request has to be kept relatively low, with 100 being a reasonable upper end. As mentioned before, due to this merging and discarding, it is also not possible to offer pagination of the search results, i.e. to enable the user to see "page 2" and so on of the result. Getting the result on any given page would require persistent caching of all the results in the federated service or to retrieve all previous pages dynamically, which would be very wasteful computationally and slow. In practice the user is better served by refining the search than by browsing multiple pages.

Establishing the Score

A relevance score can be established using a method called *term frequency – inverse document frequency*, TF-IDF for short (https://en.wikipedia.org/wiki/Tf-idf). TF-IDF assigns a unique weight to each pair (term, document) where term is a properly pre-processed word extracted from the document itself. The weight is zero if the term is not present in the document, therefore is not relevant. On the contrary, the weight is one if the term has the highest relevance in the document.
To compute the IDF portion of the weights, TF-IDF method requires access to all the documents in the corpus. However, the PaNOSC federated search relies on each facility to store their set of documents shared, compute and provide properly scored search results. As a consequence, the PaNOSC federated search does not have access to the full set of documents, so this method cannot be used unless the infrastructure is re-designed and re-implemented, which is outside of the scope of this project.

A solution to this issue was found in the publication "TF-IDuF: A Novel Term-Weighting Scheme for User Modelling based on Users' Personal Document Collections"(https://www.gipp.com/wp-content/papercite-data/pdf/beel17.pdf). The relevant finding in this paper is that the scoring results performed with TF-IDF are minimally impacted if you perform the scoring independently on disjoint subsets of the complete set of documents. If the disjoint set of documents is defined as the set of documents that each facility has available in their catalogue and it is not accessible by another facility, we can apply the process highlighted in the paper, compute the scores locally and compare them globally. Therefore, we decided to adopt the TF-IDuF method. Scores are computed independently at each facility utilising the TF-IDF weights computed locally, as indicated to the paper above. The results are then merged at the federated level and sorted according to the externally computed scores.

Scoring Algorithm and Process

The scoring process happens in three phases:

- populate scoring information
- weight computation
- score computation.

Populating scoring information

This step is necessary to provide to the scoring service the key information to compute the query score on.

Each facility has to retrieve from their catalogue system all the entries that are publicly accessible in PaNOSC. Next, each entry needs to be filtered and reformatted according to the relevant fields list. This list explicitly selects which entries' fields should be transferred to the scoring system as scoring information.

The minimal viable set of relevant fields is the ones containing exactly the fields provided to the federated search, although each facility has the freedom to enrich the set with additional fields which may provide more insight in the content and improve the accuracy of the relevancy

score.
The final step is to insert the scoring information in the scoring system using the Item endpoint.
Figure 6 illustrates the details of the complete workflow from catalogue entries to terms weights. The top three blocks illustrate what has been discussed in this paragraph: extracting entries from catalogue, selecting the scoring information and inserting them in the scoring system.

Weights computation

Each item inserted in the scoring system, is pre-processed to extract meaningful terms (Fig. 6, block titled “Terms”). As a result of the weight computation, each term is assigned a weight for each item where it is found (Fig. 6, block named “Weights”).

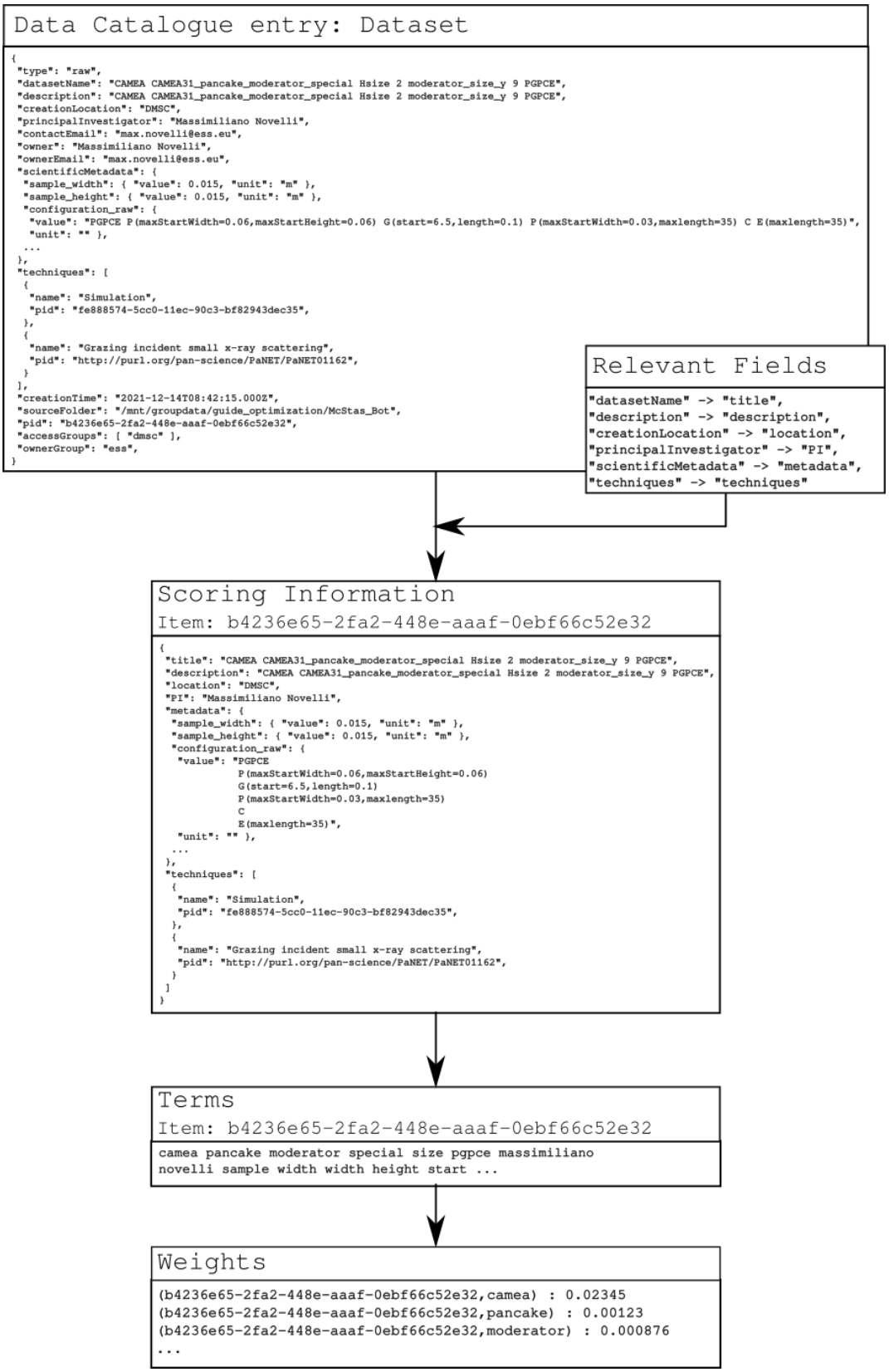


Figure 6: Workflow to populate scoring service: From catalogue entries to terms weights

To further clarify, the scoring system will store one weight for each pair (item, term). Its value is obtained according to the TF-IDF method. These weights are intended to reflect how relevant the associated term is to the item within the corpus. The output of this step is a sparse matrix with as many columns as many unique terms have been extracted from the whole set of items and as many rows as many items have been inserted in the scoring system. The weight matrix must be saved within the scoring system, possibly, in a dedicated database as part of the weight computation process.

The extraction workflow implements the following series of steps in the exact order on each item:

1. retrieve item's scoring information
2. stringify scoring information
3. convert text to lower case
4. remove punctuation (pass 1)
5. remove stop words (pass 1)
6. remove apostrophes
7. remove extra spaces
8. remove stop words (pass 2)
9. extract words from each item
10. applying stemming to all words and create terms
11. remove punctuation (pass 2)
12. remove spaces (pass 2)
13. remove short terms (terms with length 1)
14. computes TF for each term of each items
15. computes IDF for each term
16. computes TF-IDF for each (term, item) pair

This is a resource intensive step. It can be performed off-line at reasonable intervals. The weight computation needs to be re-run after updating or adding new items in the corpus, especially when the new information would change the balance of the weights. The decision when the weights computation should be run is left to the individual facility.

Score computation

Users submit a query q_u and the number of items requested n_u to the federated search, which in turn, it submits the query as-is to each facility search API. Each API has to retrieve the results set from their catalogue, and then submit it together with the query to the scoring system.
The scoring system will perform term extraction on the query, extracting meaningful terms needed for the scoring. The weights matrix is reduced by selecting only the rows related to the items present in the results set and the columns matching the query's meaningful terms. Missing values are set to 0 by default. Items' scores are computed performing a comparison of the query's weights and items' weights.

The items in the results set are matched with their relevance score, sorted according to the score, the first n_u are sent back to the federated search, where n_u is the number of items requested by the user. If the user does not provide the number of items requested n_u, the system imposes by default a limit n_d which can be set in the configuration.

Scoring Implementation

A reference implementation of the scoring system and all its subsystems has been done in Python which, at this point in time, offers the best set of libraries for natural language processing (NLP for short). NLP libraries are used to extract meaningful terms from each item's scoring information populated in the scoring system. We refer to this process as item pre-processing, or terms extraction or, simply, extraction.

The code for the PaNOSC Search Scoring is accessible in the following repository:

https://github.com/panosc-eu/panosc-search-scoring

The repository contains all the necessary code and documentation on how to run the service locally, or in a docker container. It also explains how to populate the service and provides jupyter notebooks with example code on how to integrate SciCat or any other catalogue system with the scoring service itself.

Scoring service

Figure 7 provides a detailed overview of how the PaNOSC Search Scoring (PSS) system is implemented and the endpoints groups available through the API.

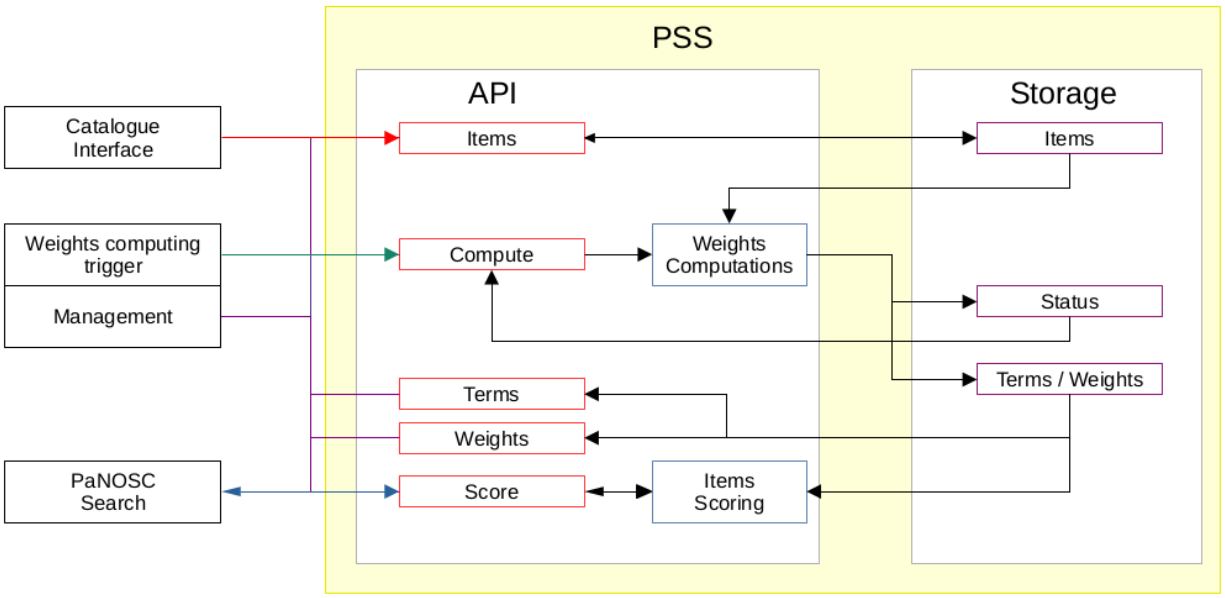


Figure 7. Search Scoring System

The implementation of the PaNOSC Search Scoring is available as an open source project at the repository https://github.com/panosc-eu/panosc-search-scoring under the master branch. At the time of writing this deliverable the latest version is v1.0-beta-4. A docker image tagged accordingly is available on https://hub.docker.com/repository/docker/nitrosx71/panosc-search-scoring.

If a facility decides to deploy the reference implementation of the scoring, they can do so by leveraging the docker-compose available in the official repository which deploys the latest release of the service together with a mongodb which is used as permanent storage for scoring information, and scoring weights.

Common Search API Endpoint Implementations

Reference API

The reference implementation has been updated to include all the changes highlighted in the previous sections. The code for the reference implementation of the API is available in this repository:

https://github.com/panosc-eu/search-api

The master branch is maintained in a ready-to-run state and can be used to deploy it locally for testing purposes or as a base to develop additional functionalities.

In order to install it and run it locally, please follow these steps:

- `git clone https://github.com/panosc-eu/search-api.git`
- `cd search-api`
- `npm install`
- set environmental variables (please read below for more info)
- `npm start`

The service will be accessible at http://localhost:3000/api. To test the reference implementation with the technique's expansion enabled, they have to define the environmental variable PANET_BASE_URL with the URL of their instance of the technique service mentioned previously.

ICAT

The ICAT community has developed an implementation of the PaNOSC search API (the ICAT search API) for use with facilities that are running the ICAT data catalogue. The code can be found at https://github.com/ral-facilities/datagateway-api. It is a Flask-based application that fetches data from an ICAT instance https://flask.palletsprojects.com. At the time of this writing, a first version that is ready to be used and tested by other facilities has been released. The minimum required prerequisites for the ICAT search API are a Python 3.6 installation, an ICAT server (https://github.com/icatproject/icat.server) and the anonymous ICAT plugin authenticator (https://github.com/icatproject/authn.anon).

The configuration file is called config.json (of which an example "config.json.example" is given in the source tree). In the configuration file the "datagateway_api" JSON object is of no interest for the ICAT search API and can be safely removed. Notable fields are "log_location" that as the name says points to the log file location, "search_api.extension" that is the base path of the ICAT search API and "search_api.icat_url" that is the URL of the ICAT instance from where data

must be fetched.

The ICAT search API outputs data in the format defined by the PaNOSC data model. To interface with ICAT, there needs to be a way of translating between this data model and the ICAT schema.
To map between each data model, a JSON file called search_api_mapping.json is used (the source tree contains an example named "search_api_mapping.json.example") which defines the mappings for each PaNOSC entity (and all the attributes within them). This is configurable so these mappings can be changed as needed.

Within the mapping file, each of the JSON objects represents a PaNOSC entity. Inside each object, there is a base_icat_entity which defines which ICAT entity the PaNOSC entity links to. There are also key-value pairs of all of the fields which exist for the PaNOSC entity, where the value is the ICAT field name. For fields which are related entities, the value contains a JSON object instead of a string. The contents of this object are the PaNOSC entity name that the field name relates to and also the ICAT field name translation.

Some entities including Affiliation and Technique are going to be added in the new ICAT 5 release (see: https://github.com/icatproject/icat.server/). Such entities do not exist in previous ICAT versions. Please refer to ICAT documentation for the list of entities.

There is one piece of functionality that has not been implemented yet: units and prefixes.

To run the ICAT search API using the Flask built-in development server, users should follow these steps:

1. git clone https://github.com/ral-facilities/datagateway-api
2. cd datagateway-api/datagateway_api
3. cp config.json.example config.json
4. edit config.json as needed to match user environment
5. cp search_api_mapping.json.example search_api_mapping.json
6. edit search_api_mapping.json as needed to match the way user uses ICAT
7. curl -sSL https://raw.githubusercontent.com/python-poetry/poetry/master/get-poetry.py | python -
8. source ~/.poetry/env
9. poetry install
10. poetry run python -m datagateway_api.src.main

By default the ICAT search API runs on http://localhost:5000.

The following queries retrieve the number of instruments and a possible instrument called "Test Instrument" (assuming to have set "search-api" as the base path of the ICAT search API in the configuration file config.json):

```
http://127.0.0.1:5000/search-api/instruments/count
http://127.0.0.1:5000/search-api/instruments?filter=
      {"where":{"name":"Test Instrument"}}
```

Do not use the Flask built-in development server when deploying to production. It is intended for use only during local development. It is

not designed to be particularly efficient, stable or secure.
For a production deployment use a WSGI container. An example of a production deployment is given using the WSGI container uWSGI in conjunction with the Nginx web server.

Install the WSGI container uWSGI with the command:

```
pip install uwsgi
```

To install all the Python dependencies, create a file **requirements.txt** in the root folder of the ICAT search API with the following content:

```
Flask-RESTful==0.3.9
SQLAlchemy==1.3.8
PyMySQL==1.0.2
Flask-Cors==3.0.9
apispec==3.3.0
flask-swagger-ui==3.25.0
PyYAML==5.4
python-icat==0.21.0
suds-community==0.8.4
py-object-pool==1.1
cachetools==4.2.1
Flask-SQLAlchemy==2.4.4
requests==2.25.1
python-dateutil==2.8.1
pydantic==1.8.2
```

and execute the command:

```
pip install -r requirements.txt
```

In the root folder of the ICAT search API create a file uwsgi.ini with the following content:

```
[uwsgi]
master=true
single-interpreter=true
enable-threads=true
plugin=python3
socket=/tmp/search-api.sock
chmod-socket=666
mount=/search-api=datagateway_api/wsgi.py
```

To run the application execute the following command inside the root folder of the ICAT search API:

```
uwsgi --ini uwsgi.ini
```

Install the Nginx web server (assuming to have a Debian/Ubuntu system):

```
sudo apt-get install nginx
```

Inside the folder **/etc/nginx/sites-available** create a file called **flaskconfig** with the following content:

```
server {
    listen 80;
```

```
        server_name localhost;

        location / {
            include uwsgi_params;
            uwsgi_pass unix:///tmp/search-api.sock;
        }
    }
```

Execute the following commands:

```
cd /etc/nginx/sites-enabled
sudo ln -s ../sites-available/flaskconfig .
sudo service nginx restart
```

The ICAT search API will be available at http://localhost/search-api. The use of the Apache web server with the mod_wsgi could be an alternative. To see other production deployment options please visit https://flask.palletsprojects.com/en/2.0.x/deploying/.

SciCat implementation

The code for the Search API SciCat implementation is accessible in this repository:

https://github.com/SciCatProject/panosc-search-api

The master branch is maintained in a ready-to-run state. Facilities that would like to deploy it on their infrastructure, can just download the latest commit from the master branch. We are planning to start creating official releases properly tagged. At the time of this writing, the SciCat implementation is complete with advanced technique querying and search scoring integration.

To run a local copy of the search, users should follow these steps:

1. **git clone https://github.com/SciCatProject/panosc-search-api.git**
2. **cd panosc-search-api**
3. **npm install**
4. **set environmental variables (please read below for more info)**
5. **npm start**

In order to simplify the service configuration, a template file https://github.com/SciCatProject/panosc-search-api/blob/master/local_test.bash.template has been added to the repository which encompasses steps 4 and 5. Please make a copy of it, rename it local_test.bash, assign the correct values to the variables as explained below and use it to run the search api locally.

The environmental variables required to run the search api locally are the following:

- BASE_URL : base url of SciCat backend
- FACILITY : name of your facility
- PSS_BASE_URL : base url of the PaNOSC Scoring system
- PSS_ENABLE : 1 if PaNOSC scoring system is enabled, 0 if not
- PANET_BASE_URL: base url of the Techniques Pan Ontologies API

If you want to run the search API in a docker container, you can build the docker image locally with the command:

```
docker build --tag panosc-search-api:local .
```

This command uses the docker file provided in the repo. Users should make sure to configure the environment variables to match their IT infrastructure, before creating the container.

Invenio RDM implementation

Invenio RDM is a turn-key research data management repository platform based on Invenio Framework and Zenodo.

The search backend of Invenio RDM is ElasticSearch and the PaNOSC search API implementation to this data catalogue type is based on the ElasticSearch connector of the Loopback framework.

The metadata model of Invenio RDM is an extension of the DataCite metadata scheme. The API implementation maps the elements of this metadata scheme to the model of the PaNOSC search API.

Access of the files in the Invenio RDM is implemented in the provider with the record API of the Invenio RDM.

The code for this implementation is available in this repository:

```
https://github.com/panosc-eu/search-api/tree/dev/invenio-rdm-provider
```

The installation of a local instance of the provider is very similar to the installation of other PaNOSC search API implementations and it consists the following steps:

1. git clone –branch dev/invenio-rdm-provider https://github.com/panosc-eu/search-api.git
2. cd search-api
3. npm install
4. set INVENIO_IP environment variable to the address of the InvenioRDM instance
   (this is necessary for file access)
5. set the address of the ElasticSearch service in the server/datasources.json file
6. npm start

The creation of a DockerImage to run this search API provider in a containerized environment is identical to the solution defined in SciCat implementation.

Status of Endpoints at Partners

The list of facilities providing data to a running instance of the federated API is provided in the browser at the root URL where the API is running. The current deployment at ESS can be reached at the URL:

https://federated.scicat.ess.eu/

This URL provides information on the configuration of the instance running. At the time of writing this deliverable, the following JSON object is returned:


```
{
 "uptime_seconds" : 6646268.156,
 "uptime" : "1846:11:08",
 "api_version" : "v2.2",
 "docker_image_version" : "v2.2",
 "hosting_facility" : "ESS",
 "environment" : "production",
 "data_providers" : [
  "https://icatplus.esrf.fr/api",
  "https://scicat.ess.eu/panosc-api",
  "https://fairdata.ill.fr/fairdata/api",
  "https://searchapi.maxiv.lu.se/api"
 ]
}
```


As we can see from the information provided, the ESS instance of the federated search API pulls data directly from search APIs running on ESS, ILL, ESRF, and MaxIV.

The up to date list of active data providers can be found in the following configuration file in the Github repository:
https://github.com/panosc-eu/panosc-federated-search-service/blob/master/.env

This allows anyone to checkout the latest commit on the master branch and run a fully functional instance of the PaNOSC federated search connected to all active data sources locally on their machine.

Note:
Facilities wanting to join the community of facilities that make data open via the PaNOSC search API are welcome to submit a pull request to the above file, adding their endpoint URL. This will initiate a (manual) review of their endpoint compliance before a merge.

ESS

The ESS focus has been on leading the work to deploy the first version of federated search and its testing. Currently the federated API is deployed only on ESS infrastructure and is accessible at the URL https://federated.scicat.ess.eu
The reference and SciCat implementations of the PaNOSC search API have been maintained and kept in sync. Since the last deliverable, the scoring service has been developed, and the techniques service, developed by PSI under the ExPANDS project, has been integrated. The local instance of the PaNOSC search API instance, accessible at https://scicat.ess.eu/panosc-api, offers full integration with the scoring and techniques services.
For the scoring system, ESS organised a workshop to support other facilities in the adoption. Partners have received support regarding the federated search API, local search API and scoring system to different

facilities, including compliance review reports.

XFEL

European XFEL (https://www.xfel.eu/), has 6 scientific instruments, which are performing experiments since late 2017. Their metadata are stored in MyMdC (https://www.xfel.eu/), the metadata catalogue used within European XFEL.

The data and metadata information are made available to the experiment team immediately after data is taken, MyMdC used to store the metadata and provide the necessary access to the data files. In accordance with the European XFEL Data Policy (https://www.xfel.eu/users/experiment_support/policies/scientific_data_policy/index_eng.html), data and metadata have a default embargo period of 3 years, during which access is restricted to the experimental team. After the embargo period data and metadata became public, being possible for external people to request access to it.

European XFEL metadata provides RESTful APIs that allow their metadata to be queried, including the "Search API" defined within WP3. The current list of implemented methods can be found at https://in.xfel.eu/metadata/api-docs/index.html (default available APIs) and https://in.xfel.eu/metadata/api-docs/index.html?urls.primaryName=PaNOSC%20API%20Docs (PaNOSC search APIs). This endpoint (https://in.xfel.eu/metadata/api) currently serves metadata related to all Open Data (and embargoed) experiments, provided that valid authentication is provided.

In order to fulfil the PaNOSC search API, XFEL did integrate on MyMdC the techniques defined on project https://expands-eu.github.io/ExPaNDS-experimental-techniques-ontology, however it is responsibilities of the Instrument Experts and users to select them and correctly assign it to the taken data. MyMdC endpoint is fully compliant with the federated search, except for the scoring implementation which is, at the time of publishing of this document, under development.

Using MyMdC any registered user can query live data from all the proposals (s)he has access to. The Federated Search, being registered as a trusted to the endpoint, has access to all the publicly accessible data.

ESRF

There are more than 40 beamlines currently performing experiments at the ESRF. Their data and metadata are archived and stored in a metadata catalogue named ICAT (https://icatproject.org/). Following the ESRF data policy (http://www.esrf.eu/datapolicy), the data is made available as soon as the data is produced and accessible via the data portal (https://data.esrf.fr). Data and metadata will be publicly available after the embargo period (3 years by default) during which access is restricted to the experimental team, represented by the Principal Investigator (PI). Each investigation has its own persistent identifier (DOI) that is obtained from Datacite (https://datacite.org/). The data of the Human Organ Atlas project (https://human-organ-atlas.esrf.fr/) has been made accessible from the search API (cf. Improvements and documentation updates).

An OAI-PMH endpoint has been developed and provides access to the high level metadata of all the investigations done at the ESRF (https://icatplus.esrf.fr/oaipmh). Besides, a search API has been deployed inline with the PaNOSC deliverable that exposes a subset of the public data (https://icatplus.esrf.fr/api/datasets).

The local catalogue (https://data.esrf.fr) authentication uses openID. It allows users and staff to access their data. Nevertheless, the search API allows anonymous access. It is also possible to create an account via the ESRF User portal (https://smis.esrf.fr/). Every user that participates in an experiment has a role. The main roles are: Principal investigator, local contact, scientist, participant, etc. The PI of an experiment can add a user as a collaborator. This allows sharing the data (https://www.youtube.com/watch?v=FDUFPpnllxE).

More than 900 parameters have been identified as metadata (https://gitlab.esrf.fr/icat/hdf5-master-config/-/blob/master/hdf5_cfg.xml), and are captured and stored automatically during the data acquisition. One of these parameters is the technique that we foreseen to make mandatory. We are currently using the search API provided as a demonstrator, however we are in the process of deploying the search API implemented within ICAT.

ILL

The data catalogue of the ILL (https://data.ill.fr) provides access to both embargoed and open data. A search interface allows users to obtain metadata concerning proposals based on proposal ID, instrument, reactor cycle and also more open, full-text searches. We use DataCite as the registration agency of DOIs for the data and the data catalogue has been registered with Re3Data since 2016 (https://www.re3data.org/repository/r3d100012072).

The ILL provides an OAI-PMH endpoint that enables harvesting of metadata of the available open data (https://data.ill.fr/openaire/oai). Currently metadata concerning more than 2500 proposals since 2013 is available through this endpoint. The endpoint has been registered with OpenAire since April 2021.

Concerning the Search API of WP3, a production version has been in operation since May 2021 (https://data.ill.fr/fairdata/api). This endpoint serves metadata related to all proposals that have Open Data (currently more than 2500).

CERIC-ERIC

As part of the current deliverable the ICAT data catalogue has been fully deployed in production at CERIC replacing the previous test deployment that was using cloud resources provided by the German Electron Synchrotron DESY. The current CERIC deployment of the ICAT data catalogue can be accessed at https://data.ceric-eric.eu.

All the objectives set in the previous deliverable have been achieved. The OAI-PMH service previously implemented at CERIC has been replaced with the one provided by the ICAT community (https://github.com/icatproject/icat.oaipmh) and is accessible at

https://data.ceric-eric.eu/oaipmh/request?verb=Identify. The default implementation of the WP3 search API has been replaced with the one implemented by the ICAT community and its base URL is https://data.ceric-eric.eu/search-api. Both the OAI-PMH service and the search API are disseminating real data.

A metadata ingestion service, python based, has been developed and deployed. This service is in charge of populating ICAT catalogue with metadata coming from user office platform (currently VUO) and from the data file collected during the experiments and saved on the Elettra storage system. The software is modular in such a way to be adapted to other "sources" (e.g. other storage systems) of metadata and more types of "destinations" (e.g. others catalogues). Thanks to its modularity, it is easy to add capabilities to extract metadata from new file formats as well.

At the beginning of 2022 the number of open datasets has increased. This is due to fast track and Covid-19 related experiments which have a shorter embargo period, these will be immediately searchable using PaNOSC interfaces. Due to the distributed nature of the CERIC consortium, a peculiar goal is to collect all the data acquired during experiments all over the partner facilities. CERIC had a successful pilot of transferring data between TU Graz labs and the Elettra storage system.

ELI

The Extreme Light Infrastructure (ELI) consists of complementary facilities located in the Czech Republic, Hungary and Romania. The ELI facilities, built as individual construction projects, are now coming together as an integrated organisation, the ELI European Research Infrastructure Consortium (ELI ERIC), that will be in charge of their joint operations.

ELI is still in the evaluation phase of selecting the proper data catalogue. A test ICAT instance and a test Invenio RDM instance are deployed and both of these data catalogue solutions support the PaNOSC search API and the OAI-PMH technologies. ELI facilities do not have available open data to supply the Federated Search API, however demo metadata is available on the Federated Search API based on the previously delivered json based data file.

The Federated Search Service

Backend API

The PaNOSC Federated Search API can be deployed in any suitable cloud or on-premise location. It has not been designed with a particular hosting organisation in mind. Given its stateless implementation, multiple instances can exist completely independently without the need to sync any information between them. The code for the Federate Search API is accessible in the following repository: https://github.com/panosc-eu/panosc-federated-search-service

The production branch is named master. Official versions are also marked with tags. Current release tag at the time of writing is v2.5. For more information about how to run the federated search both locally or in a docker container, please check the Readme file and all the documentation contained in the repository. The repository also provides a test docker compose to run the same service in a docker container.

The repository also contains a bash script to run the service locally, named "run_locally.bash". This script also leverages the official list of data providers defined in the file:

https://github.com/panosc-eu/panosc-federated-search-service/blob/master/.env

Federated items selection

The federated search API collects the $n_{f(i)}$ results from each facility *i*. Results $n_{f(i)}$ returned by facility *i* are less than or equal to the number of results requested by the user n_u. These results will be provided with an associated relevancy score computed according to the TF-IDuF methodology. The relevancy score will have a value between 0 and 1, where 1 indicates the most relevant results.
All the results will be combined together and sorted in descending order according to the relevancy score. The first n_u results are selected as the most relevant items according to the query received, and sent back to the user as query results. n_u is the number of items requested by the user.

Deployment Procedure

The git repos, providing the code for the federated search, also contains multiple options to run it locally.
First clone the repo locally:

- `git clone https://github.com/panosc-eu/panosc-federated-search-service.git`
- `cd panosc-federated-search-service`

To run an instance locally with the local code but with the official data providers, please the script provided:

- `./run_locally.bash`

If you would like to build a docker image for your own use, you can run the following docker command:

- docker build
  -t my-local-federated-search:v1.0
  -f ./search-api/Dockerfile
  ./search-api

A script to facilitate the process to publish the docker image is included in the repository. It is named docker-image-release.sh and it is found in the main folder. This script is used to upload and publish the official docker images, which can be achieved with the following command:

- ./docker-image-release.sh <docker-hub-user> <git-commit-or-tag>

If we want to publish the docker image for the new release 2.2 under the ess account, this will be the command to run:

- ./docker-image-release.sh ess v2.2

To facilitate testing of the federated search two docker compose files have been included in the repository:

- ./test/docker-compose-local-data-provider.yaml
  This file starts a full stack of containers and allows to test the federated search locally in an completely isolated environment. The data providers are included and also run locally in containers also
- ./test/docker-compose-local-data-provider.yaml
  This file starts a single docker container with the federated search configured to use the official list of data providers

In both cases, the federated search API will be available at the local URL http://localhost:3000/api or through the explorer UI: http://localhost:3000/explorer.

Configuration of the federated search is achieved setting the following environmental variables:

- API_VERSION : API version used in the running instance. Default: unknown
- DOCKER_IMAGE_VERSION : tag of the docker image used for the running instance, if any. Default: unknown
- HOSTING_FACILITY : name of the hosting facility for the running instance. Default: unknown
- ENVIRONMENT : string identifying the environment where the instance is running. Example: develop, test, or production. Default: unknown
- PROVIDERS : comma separated list of facilities PaNOSC local search apis that are used when running queries. Example: "https://icatplus.esrf.fr/api,https://scicat.ess.eu/panosc-api,https://fairdata.ill.fr/fairdata/api". Default: unknown
- DEFAULT_LIMIT : number of results returned from each facility if no limit is provided in the filter parameter. Example: if set to 10. When no limit is provided in the filter, the federated search will

return 10 results from each facility. Default: 100

Those variables are set automatically in the scripts highlighted above.

As we already mentioned before, an instance of the PaNOSC federated search is currently running at ESS. The main URL is https://federated.scicat.ess.eu/, which provides the status of the service in json format, including the list of data providers.
The explorer interface is accessible at https://federated.scicat.ess.eu/explorer/, while the API base URL is https://federated.scicat.ess.eu/api.

Web Frontend

The formal deliverable of the Search Service is fulfilled with the availability of an implementation of the Search API that federates results from the participating partner institutes. To make the search facility more accessible to end users a web frontend (or another kind of end user interface) is required. This is also available from the collaboration with WP4, where a user interface based on former PaN Portal's frontend is being developed.

The current prototype, at the time of writing available at http://pan-dev-portal.eli-laser.eu, helps the user to formulate complex Search API queries. Overall design and features are yet to be refined. The codebase can be found at https://github.com/panosc-portal/searchui

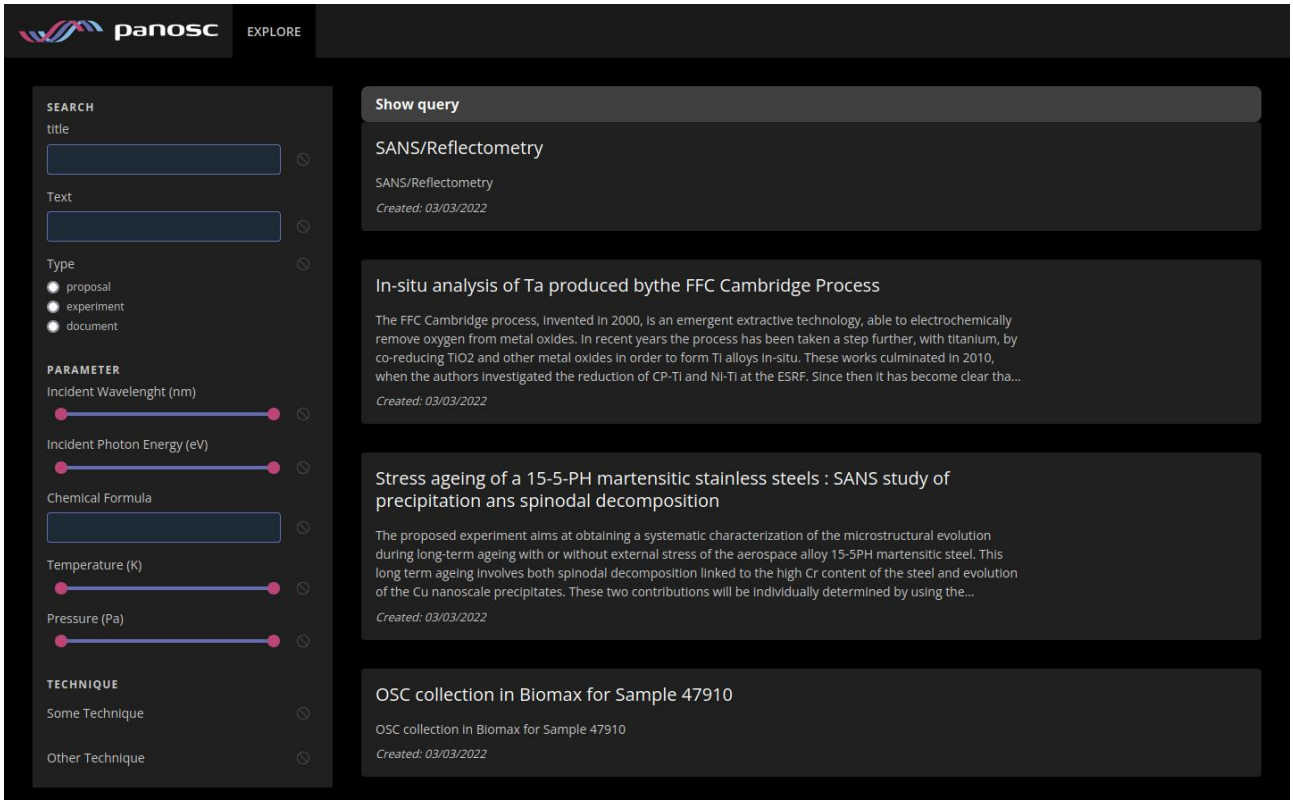


Figure 8: Screenshot of the federated search demonstrator web frontend.

Summary and Outlook

In the Data Catalogue Work Package (WP3), this deliverable marks an important achievement. The remaining deliverables in the WP are focussed on metadata quality and the local implementation of the data catalogues (OAI-PMH harvesting, integration of data sources, search endpoints and compliance with API definition, etc.) This deliverable defines the functionality baseline for the PaNOSC Domain Specific Search as one of the core pillars of our open/FAIR data infrastructure. The partners now share a better understanding of data curation tasks, dataset taxonomies and the types of queries on the data repositories that might be relevant. Following the deployment of the Federated Search Service and the associated Web Frontend, we have seen significant momentum to add more features and search functionality. This corroborates the usefulness of the iterative development approach that has been taken. Before more features will be considered, though, we need to ensure there are a significant number of compliant implementations of the common API in its most up to date version. The quality of the search results is partially depending on the correct implementation of the search API and the scoring. The metadata corpus held in the local data catalogues needs to follow the best practices and agreed mappings for queries involving experimental technique, parameters, roles, etc.

In terms of software development tasks, we expect to be able to resolve a number of smaller issues in the common code base as they arise during the course of the PaNOSC project and for the coming future. Operational stability and fault tolerance are important aspects that have not had much attention in the proof of concept operations so far. Moving forward, the main tasks should be to ensure the local catalogues have a fully compliant connection to the federated search and the local curation of metadata is in a good shape at least for new datasets entering the repositories.

Appendix 1 Search Query Examples

Leveraging the ESS instance of the federated search api, we can provide live queries examples that users can test directly from their machine.

Example 1: Datasets where title contains the word data

This query search for all the datasets where the title contains the word *data*, order the results alphabetically and then returns the first 10

Endpoint: Datasets

Filter

```
{
    "limit": 10,
    "where" : {
        "title" : {
            "like" : "data"
        }
    }
}
```

REST call

```
https://localhost:3000/api/Datasets?filter=%7B%22limit%22%3A10%2C%22where%22%3A%7B%22title%22%3A%7B%22like%22%3A%22data%22%7D%7D%7D
```

Response


```
[
  {
    "pid": "20.500.12269/00fd159a-0fe7-46d2-855d-a5dfa3d4dc13",
    "title": "V20 data",
    "isPublic": true,
    "size": 0,
    "creationDate": "2019-05-31T17:56:48.000Z",
    "score": 0,
    "provider": "https://scicat.ess.eu/panosc-api"
  },
  {
    "pid": "20.500.12269/026fe366-b469-41b7-9e5e-b5cf8b91e992",
    "title": "V20 data",
    "isPublic": true,
    "size": 0,
    "creationDate": "2019-06-01T09:36:43.000Z",
    "score": 0,
    "provider": "https://scicat.ess.eu/panosc-api"
  },
```

```
    {
      "pid": "20.500.12269/0311fead-04fb-45bd-b848-4a471fb50481",
      "title": "V20 data",
      "isPublic": true,
      "size": 0,
      "creationDate": "2019-06-01T16:00:45.000Z",
      "score": 0,
      "provider": "https://scicat.ess.eu/panosc-api"
    },
    ...
  ]
```


Notice that all the returned items have a score of 0. It is important to be aware that the scoring is triggered when the filter keyword "query" is used.

Example 2: Datasets relevant to "data beam"

This query searches for all the datasets that are relevant to the keywords *data* and *beam*, orders them by the relevancy score and returns only the best 10. This query leverages the scoring services.

Endpoint: Datasets

Filter


```
  {
      "limit":10,
      "query":"data beam"
  }
```


REST call


```
https://localhost:3000/api/Datasets?filter=%7B%22limit%22%3A10%2C%22query%22%3A%22data%20beam%22%7D
```


Response


```
  [
    {
      "pid": "20.500.12269/BRIGHTNESS/MB0033",
      "title": "Sample Data from multiblade 33",
      "isPublic": true,
      "size": 5380741,
      "creationDate": "2017-10-05T18:34:04.000Z",
      "score": 0.8508791135281516,
      "provider": "https://scicat.ess.eu/panosc-api"
    },
    {
      "pid": "20.500.12269/BRIGHTNESS/MG0031",
      "title": "Sample Data from multigrid 31",
```

```
        "isPublic": true,
        "size": 250398080,
        "creationDate": "2018-09-04T16:46:41.000Z",
        "score": 0.8508791135281516,
        "provider": "https://scicat.ess.eu/panosc-api"
      },
      {
        "pid": "20.500.12269/BRIGHTNESS/MG0023",
        "title": "Sample Data from multigrid 23",
        "isPublic": true,
        "size": 23050232,
        "creationDate": "2018-09-04T16:44:34.000Z",
        "score": 0.8508791135281516,
        "provider": "https://scicat.ess.eu/panosc-api"
      },
      ...
    ]
```


In this example, each item returned has a non-zero score. The scores computation is triggered by the keyword *query* present in the filter.

Example 3: Datasets relevant to "temperature beam"

This query searches for all the datasets that are relevant to the keywords *temperature* and *beam*, orders them by the relevancy score and returns only the best 10.

Endpoint: Datasets

Filter

```
    {
        "limit":10,
        "query":"temperature beam"
    }
```

REST call

```
https://localhost:3000/api/Datasets?filter=%7B%22limit%22%3A10%2C%22query%22%3A%22temperature%20beam%22%7D
```

Response


```
    [
      {
        "pid": "20.500.12269/3f96e58e-fc9b-11e9-b693-64006a47d649O…",
        "title": "OB-4-94cm_to_detector-SP",
        "isPublic": true,
        "size": 0,
        "creationDate": "2019-10-17T22:33:15.000Z",
        "score": 0.8685875164121688,
        "provider": "https://scicat.ess.eu/panosc-api"
```

```
    },
    {
      "pid": "20.500.12269/962ca040-c6b0-48cb-b95d-992d69f4e140…",
      "title": "Open beam WFM Slits 0.2x25",
      "isPublic": true,
      "size": 185843326,
      "creationDate": "2019-08-02T11:50:43.000Z",
      "score": 0.7963838039187,
      "provider": "https://scicat.ess.eu/panosc-api"
    },
    {
      "pid": "20.500.12269/76a55e29-21f1-4919-81c8-fbdb12ef5eac…",
      "title": "open beam,  WFM Slits 0.2x50",
      "isPublic": true,
      "size": 9913385,
      "creationDate": "2019-09-02T16:18:48.000Z",
      "score": 0.7963838039187,
      "provider": "https://scicat.ess.eu/panosc-api"
    },
    ...
  ]
```


Example 4: Documents with title containing "data"

This query search for all the documents where the title contains the word *data*, order the results alphabetically and then returns the first 10

Endpoint: Documents

Filter


```
  {
      "limit": 10,
      "where" : {
          "title" : {
              "like" : "data"
          }
      }
  }
```


REST call


```
http://localhost:3000/api/Documents?filter=%7B%22limit%22%3A10%2C%22query%22%3A%22data%20beam%22%7D
```


Response


```
  [
    {
```

```
        "pid": "10.17199/NXMV08.DSC0001",
        "isPublic": true,
        "type": "publication",
        "title": "Differential scanning calorimetry (DSC)
                  data for breast cancer cells",
        "summary": "Datasets from differential scanning calorimetry
                    (DSC) data for breast cancer cells",
        "doi": "10.17199/NXMV08.DSC0001",
        "score": 0,
        "provider": "https://scicat.ess.eu/panosc-api"
      }
    ]
```


Example 5: Datasets with title contains "open beam" including all parameters

This query returns the first 10 datasets completed by their parameters where the title contains "open beam".

Endpoint: Datasets

Filter


```
    {
        "limit":10,
        "where" : {
            "title" : { "like" : "open beam" }
        },
        "include" : [
            { "relation" : "parameters" }
        ]
    }
```


REST call


```
https://scitest.esss.lu.se/panosc-
api/Datasets?filter=%7B%22limit%22%3A10%2C%22where%22%3A%7B%22title%22%3A%7B%22li
ke%22%3A%22open%20beam%22%7D%7D%2C%22include%22%3A%5B%7B%22relation%22%3A%22param
eters%22%7D%5D%7D&access_token=%7B%22%23njs%22%3A%22eyJhbGciOiJIUzI1NiIsInR5cCI6I
kpXVCJ9.eyJfaWQiOiI2MTcxMTYzYmNjMWVjNzA5ZTA2M2M5NDgiLCJyZWFsbSI6ImxvY2FsaG9zdCIsI
nVzZXJuYW1lIjoiaW5nZXN0b3IiLCJlbWFpbCI6InNjaWNhdGluZ2VzdG9yQHlvdXIuc2l0ZSIsImVtYW
lsVmVyaWZpZWQiOnRydWUsImlhdCI6MTY0MzI4Njk2NiwiZXhwIjoxNjQzMjkwNTY2fQ.fTsNM3MKbSkj
dB0z84Wh6k5zy-
SAJ7_IlwznL3OLAH0%22%2C%22%23lb3%22%3A%22rVg991TvLx2FBcmEGcC2muFwLTZO2lhRM4KdyEHB
fKUYnvRukUTGeeFIZ5Jmv2Ei%22%7D
```


Response


```
    [
      {
```

```
    "pid": "20.500.12269/017ef494-f78e-44f3-94f8-8e584719ba3c",
    "title": "open beam,  WFM Slits 0.2x50",
    "isPublic": true,
    "size": 15195529,
    "creationDate": "2019-09-02T13:29:43.000Z",
    "score": 0,
    "parameters": [
      {
        "name": "start_time",
        "value": "2019-09-02T13:29:43",
        "unit": ""
      },
      {
        "name": "file_name",
        "value": "/data/kafka-to-nexus/nicos_00000500.hdf",
        "unit": ""
      },
      {
        "name": "title",
        "value": "open beam,  WFM Slits 0.2x50",
        "unit": ""
      },
      …
      {
        "name": "sample_description",
        "value": "V20 sample",
        "unit": ""
      },
      {
        "name": "sample_temperature",
        "value": 0,
        "unit": "celsius"
      }
    ]
  },
  ...
]
```


Example 6: Search for a specific datasets

Request dataset with pid *20.500.12269/0052f856-9615-4f9a-8575-9e180071ff32* complete with all the parameters

Endpoint: Dataset

Filter

```
{
    "include" : [
        { "relation" : "parameters" }
    ]
}
```

REST call

```
https://scitest.esss.lu.se/panosc-api/Datasets/20.500.12269%2F0052f856-9615-4f9a-8575-
9e180071ff32?filter=%7B%22include%22%3A%5B%7B%22relation%22%3A%22parameters%22%7D
%5D%7D&access_token=%7B%22%23njs%22%3A%22eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
faWQiOiI2MTcxMTYzYmNjMWVjNzA5ZTA2M2M5NDgiLCJyZWFsbSI6ImxvY2FsaG9zdCIsInVzZXJuYW1l
IjoiaW5nZXN0b3IiLCJlbWFpbCI6InNjaWNhdGluZ2VzdG9yQHlvdXIuc2l0ZSIsImVtYWlsVmVyaWZpZ
WQiOnRydWUsImlhdCI6MTY0MzI4Njk2NiwiZXhwIjoxNjQzMjkwNTY2fQ.fTsNM3MKbSkjdB0z84Wh6k5
zy-
SAJ7_IlwznL3OLAH0%22%2C%22%23lb3%22%3A%22rVg991TvLx2FBcmEGcC2muFwLTZO2lhRM4KdyEHB
fKUYnvRukUTGeeFIZ5Jmv2Ei%22%7D'
```

Response


```
{
  "pid": "20.500.12269/0052f856-9615-4f9a-8575-9e180071ff32",
  "title": "Open beam WFM Slits 0.2x25",
  "isPublic": true,
  "size": 195221294,
  "creationDate": "2019-08-02T12:03:28.000Z",
  "score": 0,
  "parameters": [
    {
      "name": "start_time",
      "value": "2019-08-02T12:03:28",
      "unit": ""
    },
    {
      "name": "file_name",
      "value": "/data/kafka-to-nexus/nicos_00000482.hdf",
      "unit": ""
    },
    …
    {
      "name": "sample_description",
      "value": "",
      "unit": ""
    },
    {
```

```
            "name": "sample_temperature",
            "value": 0,
            "unit": "celsius"
          }
        ]
      }
```

Example 7: Datasets containing specific parameters

Search for datasets containing a parameter named size with value 0 and a parameter "sample_temperature" with value 0

Endpoint: Datasets

Filter

```
      {
       "include" : [
        {
         "relation" : "parameters",
         "scope" : {
          "where" : {
           "and" : [
            {"name":"size"},
            {"value":0}
           ]
          }
         }
        },
        {
         "relation" : "parameters",
         "scope" : {
          "where" : {
           "and" : [
            { "name" : "sample_temperature" },
            { "value" : 0 }
           ]
          }
         }
        }
       ]
      }
```

REST call

https://scitest.esss.lu.se/panosc-api/Datasets?filter=%7B%22include%22%3A%5B%7B%22relation%22%3A%22parameters%22%2C%22scope%22%3A%7B%22where%22%3A%7B%22and%22%3A%5B%7B%22name%22%3A%22size%22%7D%2C

%7B%22value%22%3A0%7D%5D%7D%7D%7D%2C%7B%22relation%22%3A%22parameters%22%2C%22scope%22%3A%7B%22where%22%3A%7B%22and%22%3A%5B%7B%22name%22%3A%22sample_temperature%22%7D%2C%7B%22value%22%3A0%7D%5D%7D%7D%7D%5D%7D&access_token=%7B%22%23njs%22%3A%22eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2MTcxMTYzYmNjMWVjNzA5ZTA2M2M5NDgiLCJyZWFsbSI6ImxvY2FsaG9zdCIsInVzZXJuYW1lIjoiaW5nZXN0b3IiLCJlbWFpbCI6InNjaWNhdGluZ2VzdG9yQHlvdXIuc2l0ZSIsImVtYWlsVmVyaWZpZWQiOnRydWUsImlhdCI6MTY0MzI4Njk2NiwiZXhwIjoxNjQzMjkwNTY2fQ.fTsNM3MKbSkjdB0z84Wh6k5zy-SAJ7_IlwznL3OLAH0%22%2C%22%23lb3%22%3A%22rVg991TvLx2FBcmEGcC2muFwLTZO2lhRM4KdyEHBfKUYnvRukUTGeeFIZ5Jmv2Ei%22%7D

Response


```
[
  {
    "pid": "20.500.12269/0052f856-9615-4f9a-8575-9e180071ff32",
    "title": "Open beam WFM Slits 0.2x25",
    "isPublic": true,
    "size": 195221294,
    "creationDate": "2019-08-02T12:03:28.000Z",
    "score": 0,
    "parameters": [
      {
        "name": "start_time",
        "value": "2019-08-02T12:03:28",
        "unit": ""
      },
      {
        "name": "file_name",
        "value": "/data/kafka-to-nexus/nicos_00000482.hdf",
        "unit": ""
      },
      {
        "name": "title",
        "value": "Open beam WFM Slits 0.2x25",
        "unit": ""
      },
      {
        "name": "size",
        "value": 0,
        "unit": ""
      },
      …
      {
        "name": "sample_description",
        "value": "",
        "unit": ""
```

```
        },
        {
          "name": "sample_temperature",
          "value": 0,
          "unit": "celsius"
        }
      ]
    },
    …
]
```

Appendix 2: Techniques Ontology Examples

In this appendix, we provide more examples regarding the techniques service. We cover only the GET /techniques/pan-ontology endpoint, as it is the one to integrate in the search-API and most often used.

Example 1: Single technique expansion

Input:

```
{
  “pid” : ”http://purl.org/pan-science/PaNET/PaNET00210”
}
```

REST call:

```
/techniques/pan-ontology?where=%7B%22pid%22%3A%22http%3A%2F%2Fpurl.org%2Fpan-science%2FPaNET%2FPaNET00210%22%7D
```

Output:

```
{
  "pid” : {
     "inq” : [
       "http://purl.org/pan-science/PaNET/PaNET00210",
       "http://purl.org/pan-science/PaNET/PaNET01318",
       "http://purl.org/pan-science/PaNET/PaNET01319"
     ]
  }
}
```

Example 2: Logical or between techniques

Input:

```
{
  "or" : [
    { "pid" : "http://purl.org/pan-science/PaNET/PaNET00210" },
    { "pid" : "http://purl.org/pan-science/PaNET/PaNET00209" }
  ]
}
```

REST call:

```
/techniques/pan-ontology?where=%7B%22or%22%3A%5B%7B%22pid%22%3A%22http%3A%2F%2Fpurl.org%2Fpan-
```

```
science%2FPaNET%2FPaNET00210%22%7D%2C%20%7B%22pid%22%3A%22http%3A%2F%2Fpu
rl.org%2Fpan-science%2FPaNET%2FPaNET00209%22%7D%5D%7D
```

Output:


```
{
  "or" : [
    {
      "pid" : {
        "inq": [
          "http://purl.org/pan-science/PaNET/PaNET00210",
          "http://purl.org/pan-science/PaNET/PaNET01318",
          "http://purl.org/pan-science/PaNET/PaNET01319"
        ]
      }
    },
    {
      "pid": {
        "inq": [
          "http://purl.org/pan-science/PaNET/PaNET00209",
          "http://purl.org/pan-science/PaNET/PaNET01150",
          "http://purl.org/pan-science/PaNET/PaNET01226",
          "http://purl.org/pan-science/PaNET/PaNET01320"
        ]
      }
    }
  ]
}
```


Example 3: Logical and between techniques

Input:

```
{
  "and" : [
    { "pid" : "http://purl.org/pan-science/PaNET/PaNET00210" },
    { "pid" : "http://purl.org/pan-science/PaNET/PaNET00209" }
  ]
}
```

REST call:

```
/techniques/pan-
ontology?where=%7B%22and%22%3A%5B%0A%20%20%7B%22pid%22%3A%22http%3A%2F%2F
purl.org%2Fpan-
science%2FPaNET%2FPaNET00210%22%7D%2C%0A%20%20%7B%22pid%22%3A%22http%3A%2
F%2Fpurl.org%2Fpan-science%2FPaNET%2FPaNET00209%22%7D%0A%5D%7D%0A
```

Output:


```
{
  "and": [
```

```
      {
        "pid": {
          "inq": [
            "http://purl.org/pan-science/PaNET/PaNET00210",
            "http://purl.org/pan-science/PaNET/PaNET01318",
           "http://purl.org/pan-science/PaNET/PaNET01319"
          ]
        }
      },
      {
        "pid": {
          "inq": [
            "http://purl.org/pan-science/PaNET/PaNET00209",
            "http://purl.org/pan-science/PaNET/PaNET01150",
            "http://purl.org/pan-science/PaNET/PaNET01226",
            "http://purl.org/pan-science/PaNET/PaNET01320"
          ]
        }
      }
    ]
  }
```