EGI-Engage

Data repository for DARIAH

D6.2

Date	8 January 2016
Activity	SA2 Knowledge Commons
Lead Partner	INFN
Document Status	FINAL
Document Link	https://documents.egi.eu/document/2654

Abstract

This deliverable describes the outcome of the "Storing and Accessing DARIAH contents on EGI (SADE)" pilot application of the DARIAH Competence Centre of the EGI-Engage project. The activity developed and deployed two systems: (1) a digital repository based on gLibrary service and the EGI Federated cloud and grid infrastructure and (2) a semantic search engine. The first system helps digital humanities communities build customised and highly-available digital repositories, while the second enables the discovery and correlation of content across geographically distributed digital repositories. The presented work was developed in EGI by the EGI DARIAH Competence Centre of the EGI-Engage project.

The EGI-Engage project is co-funded by the European Union (EU) Horizon 2020 program under Grant number 654142 http://go.egi.eu/eng

COPYRIGHT NOTICE

DELIVERY SLIP

	Name	***Partner/Activity***	***Date***
From:	Giuseppe La Rocca, Roberto Barbera	EGI.eu / INFN	8/1/2016
Moderated by:	Matthew Viljoen	EGI.eu	18/1/2016
Reviewed by	Donatella Castelli Daniele Bailo	DCH-RP IT INGV	18/1/2016
Approved by:	AMB and PMB		8/2/2016

DOCUMENT LOG

Issue	***Date***	***Comment***	***Author/Partner***
V0.1	15/12/2015	First draft	G. La Rocca, Antonio Calanducci /EGI.eu
V0.2	17/12/2015	Finalised for internal review in the DARIAH CC	G. Sipos / EGI.eu-SZTAKI R. Barbera / INFN
V0.3	8/1/2016	Draft updated for external review	G. La Rocca /EGI.eu Antonio Calanducci/INFN R. Barbera/INFN
FINAL	02/1/2016	Final version	G. La Rocca/EGI.eu

TERMINOLOGY

A complete project glossary is provided at the following page: http://www.egi.eu/about/glossary/

Contents

Executive summary

This deliverable reports the status of the "Storing and Accessing DARIAH contents on EGI (SADE)" mini project, one of the three mini-projects identified in the context of the DARIAH Competence Centre (CC) of the EGI-Engage project. The project aims to raise awareness of A&H researchers of the possible benefits of using e-Infrastructure and e-Science technologies in their research programmes, create conditions for a sustained increase of the user community coming from A&H and Social Sciences as well, and increase the number of e-Science services and applications for the A&H researchers

The activities of this mini-project have been coordinated by the Italian National Institute for Nuclear Physics (INFN). At the beginning the aim of this mini-project was to create a digital repository of DARIAH contents for the A&H community. During the implementation of the work-plan the original objective has shifted a little bit to take into account the requirements of a second use case provided by the scientific community which expressed the interest to run SPARQL queries in different digital repositories. As a consequence two services, which will be presented in this report, were identified.

To implement the objectives of this mini-project the INFN, based on the experience matured during the DCH-RP FP7 EU co-funded project, updated the architecture of gLibrary, the INFN Digital Repository System, which helps scientific communities to build customised and highly-available digital repositories. The new implementation of gLibrary (refers as gLibrary 2 in the following sections) now offers the possibility to interface with digital repositories already available in different databases back-ends and exploit the EGI Federated Cloud Infrastructure to manage digital assets also in cloud Object Storages (Swift). Unfortunately at the time of writing the Austrian Academic of Science (AAS) community is currently refining the use case and the repository is under development.

In addition, during this collaboration the A&H community involved in the DARIAH CC expressed their interest in running SPARQL queries on different digital repositories and search for some potential correlations among results. To support this second use case, INFN decided to customize the architecture of the Parallel Semantic Search Engine (SSE), the service which enables the discovery and correlation of content across geographically distributed digital repositories, and to offer this service to the target A&H community. This second service has been used to develop a first demonstrative use case which is now available in production in the form of a web service. The dedicated DARIAH CC Working Group will take care to coordinate the exploitation and the dissemination of this first demonstrative use case among the A&H community.

For clarity's sake the two services:

- Will be **operated** by the INFN 24h, 7*365;
- Will be available **for use** by any member of the DARIAH community through its DARIAH-CC.

The users of the two services, which will be described in details in the next sub-sections, are represented by researchers of the A&H communities involved in the DARIAH CC project.

The outline of this deliverable is as follows: first, we introduce the use cases identified for this mini-project. Then we present the high-level architectures of the two services developed by the INFN along with the technological building blocks and their inter-dependencies. Finally, future development activities are described in the final section. The document closes with an appendix that includes the documentation of the online system.

1 Introduction

In the last decade, excellent Science has become more and more possible with e-Infrastructures. However, although various research domains, such as Medicine, Chemistry, and Physics, have already made extensive use of e-Infrastructure services, others, such as Arts and Humanities are not yet exploiting these facilities at the same pace. To bridge this gap, the EGI DARIAH Competence Centre (CC) project[1] aims to provide a wider and more efficient access to, and use of e-Infrastructures, particularly the Federated Cloud facility offered by EGI. The CC aims to make the EGI Federated Cloud more convenient to use for members of the European digital research infrastructure for the Arts and Humanities[2].

In this document we report the status of the "Storing and Accessing DARIAH contents on EGI (SADE)" pilot application that is part of the CC activities. SADE is one of the two mini-projects within the CC and aims to raise awareness of the possible benefits in using e-Infrastructure and e-Science technology in the A&H scientific domain.

For this mini-project the DARIAH contents have been provided by the Austrian Academy of Sciences[3] (AAS), one of the leading Austrian research institutions with a very long-running experience and interest in the Arts and Humanities domain. The AAS datasets represent the work on a 100+ years old collection on Bavarian dialects within the Austrian-Hungarian monarchy from the beginnings of German language to nowadays. AAS datasets are currently stored within MySQL databases without metadata and are available in different formats:

- Headwords (about 50,000 A-Z)[4];
- Records (about 40,000 plants; about 70,000 in general)[5];
- Multimedia with link to Audio-file[6];
- Multimedia with collection (about 3,000)[7];
- Multimedia connected to Headword (about 3,000)[8];
- Project specific biographies[9];
- Locations[10].

[1] https://wiki.egi.eu/wiki/Competence_centre_DARIAH
[2] https://www.egi.eu/infrastructure/cloud/
[3] www.oeaw.ac.at
[4] http://wboe.oeaw.ac.at/dboe/indices/lemma/A1
[5] http://wboe.oeaw.ac.at/dboe/beleg/142175
[6] http://wboe.oeaw.ac.at/dboe/quelle/19040
[7] http://wboe.oeaw.ac.at/dboe/quelle/18335
[8] http://wboe.oeaw.ac.at/dboe/lemma/25996
[9] http://wboe.oeaw.ac.at/dboe/indices/person/A/1
[10] http://wboe.oeaw.ac.at/dboe/indices/ort/A/1

Several data types are taken into account: text, multimedia (images, audio files etc.), URIs; as well as primary collection data, interpreted data, secondary background data and geo-data with different license opportunities.

The AAS datasets have Latin headwords connected with geo-references, names of persons who collected the data, usually a time reference and are interlinked with linked data resources to find out what is available on the internet on the certain topic.

In the plant names use case, for instance, there are many Latin headwords which can be found in the AGROVOC[11] dictionary, several geo-references in GEONAMES[12], some definitions in GERMANET[13], and some additional data resources available on Wikipedia and other wiki-projects.

To discover and correlate content in geographically distributed digital repositories the Parallel Semantic Search Engine (PSSE), developed by the INFN in the contest of the CHAIN and CHAIN-REDS[14] projects, has been extended in order to include the possibility to run SPARQL queries in parallel on some additional digital repositories (e.g. GENONAMES, DBpedia[15] and Europeana[16]) and start to introduce the support to different dictionaries (e.g. ISLEX[17] and GERMANET).

Moreover, to help the A&H community to organize AAS datasets stored in different DB back-ends and develop digital repositories of DARIAH contents on top of the EGI Cloud Storages, INFN decided to further extend the architecture of gLibrary with the Node.Js API Framework LoopBack[18]. With this new architecture of the gLibrary A&H researchers have now a powerful tool to create and organize digital repositories with datasets stored in different DB back-ends and use the Cloud technology as storage back-end to host AAS datasets. With regards to the storage back-end, cloud technologies are mature enough to improve the reliability and the availability of data. Clouds can provide the possibility of dynamic data replication among computing centres in a user-transparent way, preventing data loss as a result of localized disasters.

The Italian National Institute for Nuclear Physics (INFN) setup two different services in the pilot to demonstrate capabilities relating to the two key aspects of digital repositories:

1. Building customised and highly-available digital repositories;
2. Discover and correlate content in geographically distributed digital repositories.

The first one provides a lightweight and sophisticate tool for members of the DH community to organize and manage digital repositories on top of the EGI Federated Cloud. This system is based on gLibrary. The second one offers a Parallel Semantic Search Engine (PSSE) for the DH community

[11] http://aims.fao.org/vest-registry/vocabularies/agrovoc-multilingual-agricultural-thesaurus
[12] www.genonames
[13] www.sfs.uni-tuebingen.de/GermaNet/
[14] www.chain-project.eu
[15] http://dbpedia.org
[16] www.europeana.eu
[17] www.islex.is/
[18] http://loopback.io/

that can be used for keyword-based content discovery. In the following, we present the architecture of the two services that have been setup and tailored for the DH community.

The following tables provide a summary of the key aspects of the two systems:

Tool name	gLibrary, the INFN Digital Repository System
Tool url	Project homepage: https://glibrary.ct.infn.it API endpoint (for developers): https://glibrary.ct.infn.it:3500 (Access to be required to INFN)
Tool wiki page	https://csgf.readthedocs.org/en/latest/glibrary/docs/index.html
Description	A service to create and manage repositories of digital assets on grid, cloud and local storage
Customer of the tool	DARIAH CC
User of the service	Research groups; individual A&H researchers
User Documentation	
Technical Documentation	https://csgf.readthedocs.org/en/latest/glibrary/docs/index.html http://csgf.readthedocs.org/en/latest/glibrary/docs/glibrary2.html (APIs documentation)
Product team	INFN
License	Released under the Apache License 2.0
Source code	https://github.com/csgf/glibrary (source code)

Tool name	The Parallel Semantic Search Engine
Tool url	Current link: http://csgf.egi.eu/dariah-sse-parallel
Tool wiki page	https://csgf.readthedocs.org/en/latest/parallel-semantic-search-portlet/docs/index.html
Description	A service conceived to demonstrate the potential of Open Access Data infrastructures coupled with semantic web technologies to address the issues of data discovery and correlation
Customer of the tool	DARIAH CC
User of the service	Research groups; individual A&H researchers
User Documentation	https://csgf.readthedocs.org/en/latest/parallel-semantic-search-portlet/docs/index.html (PSSE) http://csgf.readthedocs.org/en/latest/semantic-search-portlet/docs/index.html (SSE) http://csgf.readthedocs.org/en/latest/semantic-search-

	api/docs/index.html (API SSE)
Technical Documentation	https://csgf.readthedocs.org/en/latest/parallel-semantic-search-portlet/docs/index.html
Product team	INFN
License	Released under the Apache License 2.0
Source code	https://github.com/csgf/parallel-semantic-search-portlet (PSSE) https://github.com/csgf/semantic-search-portlet (SSE) https://github.com/csgf/semantic-search-api (API)

2 Service architecture

In this section we describe the high-level service architecture of the two services, along with their dependencies, API and software requirements, that have been customized for the Arts and Humanities scientific community. The first one is gLibrary, the digital repository system developed by INFN to offer an easy-to-use service and a powerful system to handle digital assets. The second one is the Parallel Semantic Search Engine (PSSE), the service conceived to demonstrate the potential of information coupled with semantic web technologies to address the issues of data discovery and correlation. With the SSE user can search keywords in the e-Infrastructure Knowledge Base, in more than 100 languages across more than 30 million resources contained in the thousands of semantically enriched Open Access Document Repositories (OADRs) and Data Repositories (DRs). Search results are ranked according to the Ranking Web of Repositories[19]. For supporting the requirements of the Arts and Humanities scientific community a parallelised version of the SSE has been configured to simultaneously search, using different Java threads, across the above e-Infrastructure Knowledge Base and other platforms.

2.1 The gLibrary Digital Repository Service

gLibrary is the digital repository system developed by INFN. It has been used in different national and international EU-funded projects in fields such as Humanities, Earth Science and High Energy Physics. In the past months, gLibrary has evolved from a system based on Grid Storage and metadata only to a more general service, capable of handling repositories on both Grid and Cloud Storages with metadata and data records stored in both relational and/or non-relational databases. The new version of the system, called gLibrary 2.0, offers access both to existing data repositories and the creation of new ones via a simple REST API.

[19] http://repositories.webometrics.info/

For the sake of completeness, and to help reader's comprehension, below we report some terminologies used in the gLibrary lingo.

A **repository** is a virtual container of one or more data **collections**.

A **collection** provides access to data records stored on a relational DB table or to a non-relational (NoSQL) DB collection. Currently gLibrary supports MySQL, PostgreSQL, Oracle and MongoDB.

Each repository can group together one of more collections, providing a virtual and uniform interface to data tables coming from different databases that could be potentially of different types (for example one collection provides access to a PostgreSQL table and another to a MongoDB collection) and in local and/or remote servers. JSON is used as input and output data format.

Once collections are imported or created from scratch, the gLibrary REST API can be used to retrieve, create, update and delete collection's records, that in gLibrary lingo are called **items**, and to manage **replicas**. A powerful filtering system is available to make queries on collections. All criteria are specified using the query string of the REST API call. (e.g., `/v1/repos/fantasy_company/orders?filter[where][userId]=acaland&filter[where][orderQuantity][gt]=200&filter[where][limit]=100` will search for 100 orders issued by the user "acaland" with a quantity of 100).

Each item can have one or more *attachments*, which are called **replica**. Replicas can be stored on Grid Storage Elements (functionalities offered by DPM - Disk Pool Manager) and/or on Cloud Storage (OpenStack Swift is currently supported). The service offers API calls for uploading and downloading files to the configured storages, using a system based on Temporary URLs that permits direct data transfers from clients to destination storages and vice versa, without caching them on the gLibrary intermediate server.

Relations between two collections of the same repository can be created, if foreign keys are properly assigned. Currently we support one-to-many relations.

gLibrary can be useful for users that need a secure way to save and share their assets. REST API calls are authenticated using a session token based system, while authorization to repositories, collections and items can be selectively changed based on Access Control Lists (ACLs) and Roles.

Being a API based service, gLibrary can be easily integrated with other services, in particular web applications, Science Gateways, command line jobs, desktop and mobile applications.

For the previous version of gLibrary (1.0), several front-ends were available: a standalone web application[20] and several portlets deployed in a number of Liferay-based Science Gateways that offer feature to browse various repositories, download replicas, upload and register files to Grid and Cloud storages.

[20] https://glibrary.ct.infn.it

In the context of the DARIAH CC project, the Node.JS API Framework LoopBack[21] has been used to extend the architecture of gLibrary with new functionalities. Thanks to these new functionalities will be possible to import datasets already available on different back-ends and use Science Gateways and mobile applications to access to them.

2.2 gLibrary architecture with Science Gateways

gLibrary 2.0 has been rewritten from scratch and based on the Node.JS API Framework LoopBack. The high-level architecture of the gLibrary 2.0 deployed with a Science gateway or mobile device front-end is sketched in Fig.1

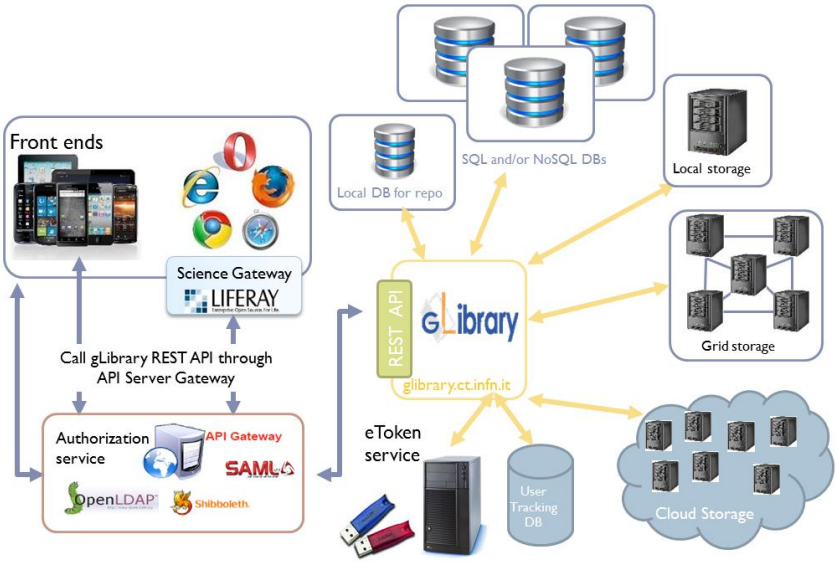

Figure 1 – The gLibrary architecture in combination with SGs

When used in combination with Science Gateways and federated authentication, an API gateway (generally deployed in the Science Gateway machine) is needed:

- The AuthN & AuthZ module: gLibrary currently supports the Federated Authentications, the traditional PKI authentications based on the adoption of X.509 certificates and based on session tokens in exchange for a username/password couple. Access to the Grid and Cloud storages are secured using the digital credentials generated on-the-fly by the eTokenServer, the standard-based solution developed by the INFN for central management of robot certificates and provisioning of proxies to get seamless and secure access to computing e-Infrastructures, based on Grid and Cloud middleware supporting X.509 standard for authorization;

[21] http://loopback.io/

- As mentioned in the previous section, several backends can be used to store metadata and data records on local storage systems, Grid Storage Elements based on DPM[22] services with a WebDAV interface and Cloud Storage (OpenStack Swift is supported). In particular, the exploitation of the EGI Federated Cloud Infrastructure will offer to the A&H community a distributed storage infrastructure accessible in a seamless and transparent way. This will contribute to increasing the resilience and the availability of the solution provided to this community.

2.3 The Parallel Semantic Search Engine High-Level Service architecture

Essential improvements for the A&H research community to dealing with RDF, XML files and SPARQL queries on different Open Access Document Repositories (OADRs), semantically enriching OADRs and Data Repositories (DRs) with a specific ontology, and to build a search engine on the related linked data to discover new science and support the reproducibility of science.

With this regard, INFN provided access to the Parallel Semantic Search Engine (PSSE) service that has been customized to address the requirements of the DARIAH CC project. In the new version of the PSSE it has been included the possibility to search across OpenAgris, PubMed and DBPedia liked data, support the integration with Altmetric[23] data and increase both speed and fault tolerance of the whole service. Next steps will be the link to dictionaries provided by the A&H community. Thanks to this service, users from the A&H research community can search in the e-Infrastructure Knowledge Base, in more than 100 languages across more than 30 million resources contained in the thousands of semantically enriched OADRs and DRs.

The multi-layered architecture of the Parallel Semantic Search Engine is presented in Fig. 2 where both the official and “de facto” Semantic Web standards and technologies[24] adopted are indicated by small logos. The first two components of the service are described below Fig. 2.

[22] www.gridpp.ac.uk/wiki/DPM_Install
[23] www.altmetric.com
[24] http://semanticweb.org/wiki/Semantic_Web_standards

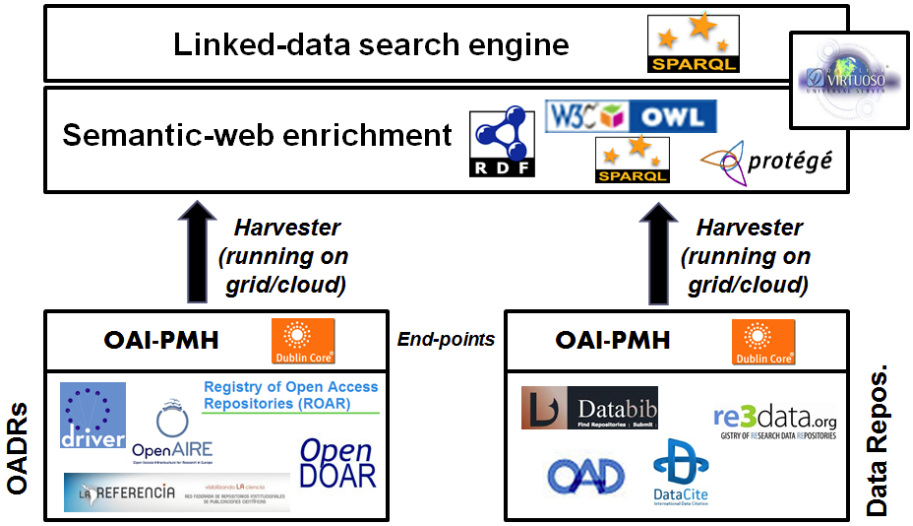

Figure 2 – Architecture of the Parallel Semantic Search Engine

2.3.1 Metadata Harvester

As shown in Fig. 2, the metadata harvester is a process running either on a Grid or on a Cloud infrastructure which consists of the following steps

1. Obtain the address of each repository publishing an OAI-PMH standard[25] endpoint;
2. Retrieve, using the OAI-PMH repository address, the related Dublin Core[26] encoded metadata in XML format;
3. Get the records from the XML files and transform the metadata in RDF format , using the Apache Jena API[27];
4. Save the RDF files into a Virtuoso[28] triple store according to an OWL-complaint ontology built using Protège[29].

2.3.2 Semantic-Web Enricher

Each RDF file retrieved and saved in the Virtuoso triple store is mapped onto a Virtuoso Graph that contains the ontology expressly deployed for the search engine, shown in Fig. 3 for the sake of completeness. The ontology, built using Dublin Core and FOAF standards, consist of:

- Classes that describe the general concepts of the domain: Resource, Author, Organisation, Repository and Dataset (where Resource is a given open access document);
- Object properties that describe the relationship among the ontology classes;

[25] www.openarchives.org/phm
[26] www.dublincore.org
[27] http://jena.apache.org
[28] http://virtuoso.openlinksw.com
[29] http://protege.standford.edu

- Data properties (or attributes) that contain the characteristics or classes' parameters.

The third, and high-level, component is the Search Engine itself, which is described in details in the next sub-section.

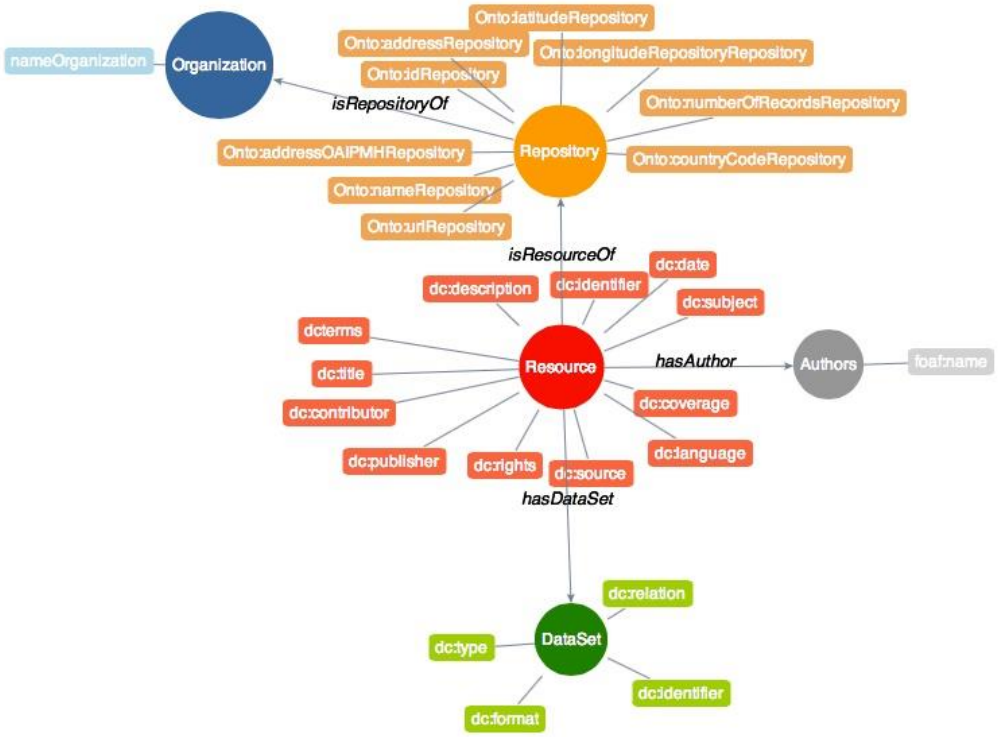

Figure 3 – Schema of the ontology used for the e-Infrastructure Knowledge Base

2.3.3 Search Engine

The home page of the Parallel Search Engine customized for the DARIAH CC project is shown in Fig. 4. Using the portlet, visitors can enter a keyword (or more keyword at the same time) and submit a SPARQL query to the Virtuoso triple store. The service is currently configured to simultaneously search across the above e-Infrastructure Knowledge Base[30], Europeana[31], Cultura Italia[32], Isidore[33], OpenAgris[34], PubMed[35] and DBpedia[36] platforms. Others repositories can easily be added since the service has been designed with modular plugins and the searches are made in parallel using different Java threads.

[30] www.chain-reds.eu/web/old-project/knowledge-base
[31] http://europeana.ontotext.com/sparql
[32] http://dati.culturaitalia.it/sparql
[33] http://rechercheisidore.fr/sparql/
[34] http://202.45.139.84:10035/catalogs/fao/repositories/agris#query
[35] http://pubmed.bio2rdf.org/sparql
[36] http://dbpedia.org/sparql

The home page of the parallel SSE[37] tailored to address the requirements of the A&H community is presented in Fig. 4.

From the text field researchers can enter a keyword and submit in parallel SPARQL queries across several digital repositories.

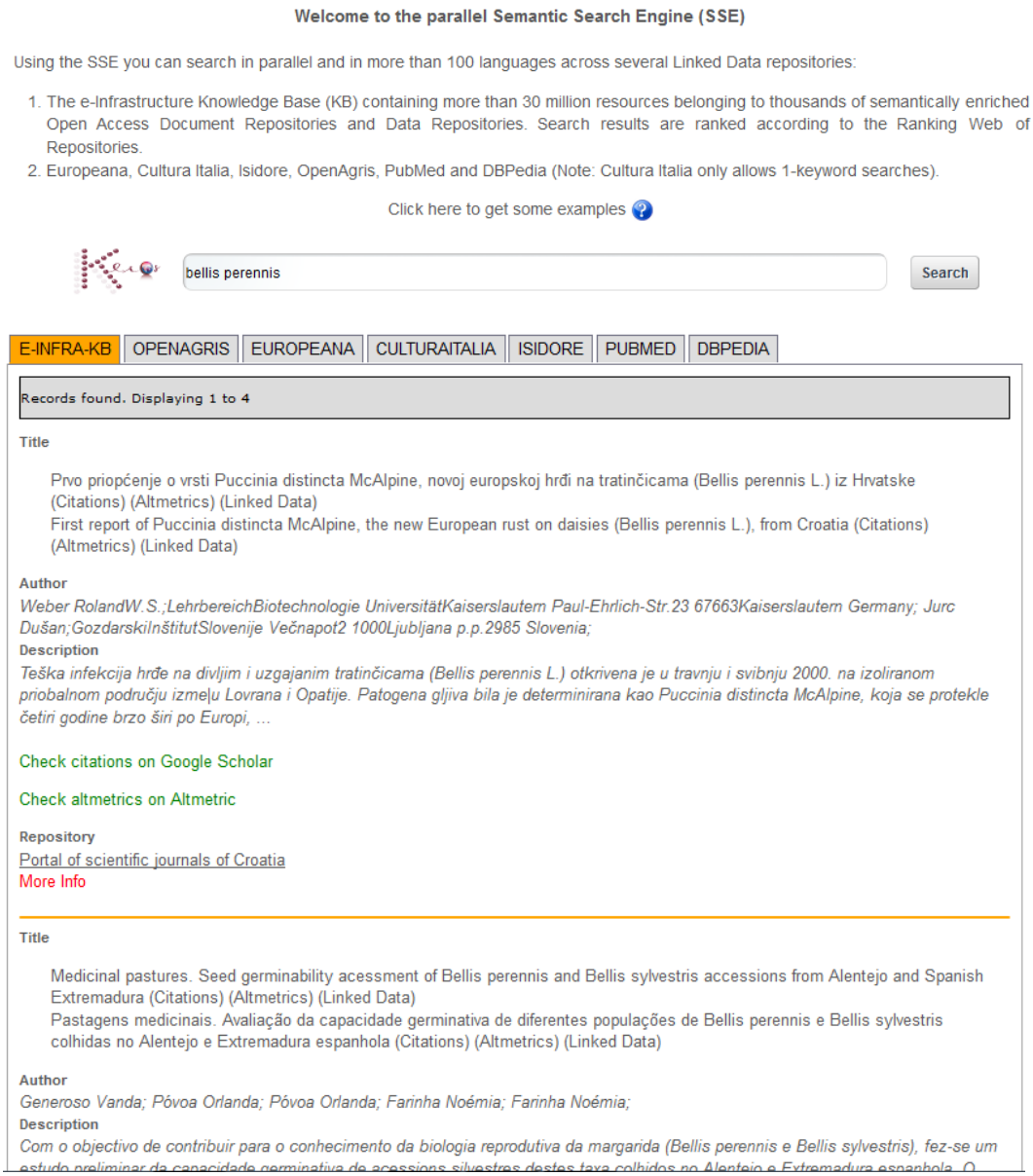

Figure 4 – Typical results of the given query

[37] http://csgf.egi.eu/dariah-sse-parallel

The results of a given query are listed in the summary view shown at the bottom part of Fig. 4. For each record found, the title, the author(s) and a short description of the corresponding resource are provided. Clicking on “More Info”, visitors can access the detailed view of the resource, as shown in Fig. 5.

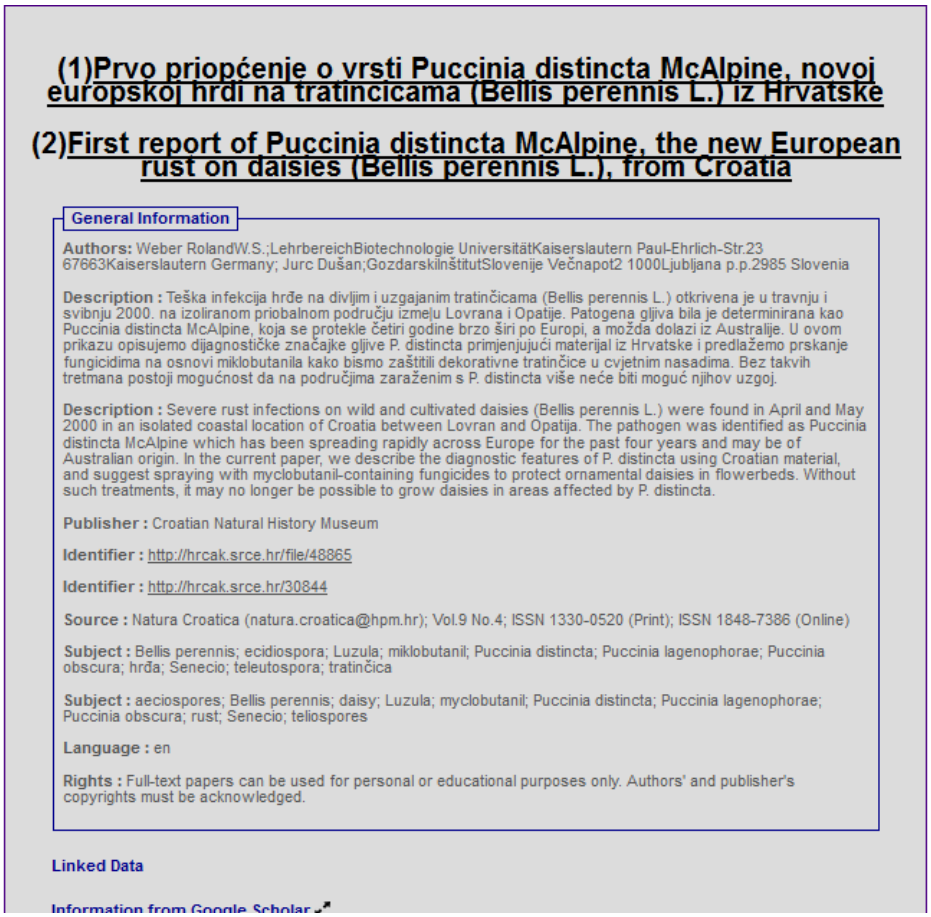

Figure 5 – Detailed view of a record found by the Parallel Semantic Search Engine

In the “Dataset information” panel users get the link to the open access document and, if existing, to the corresponding dataset. Clicking on the “Altmetrics”[38] link, it is possible to monitor the impact of the discovered result in the academia. Clicking on the “Linked data” tab, which appears at the top of the summary view, users can select one of more of the resources found and get a graphic view of the semantic connections among Authors, Subjects and Publishers, as shown in Fig. 6.

[38] http://www.altmetric.com/

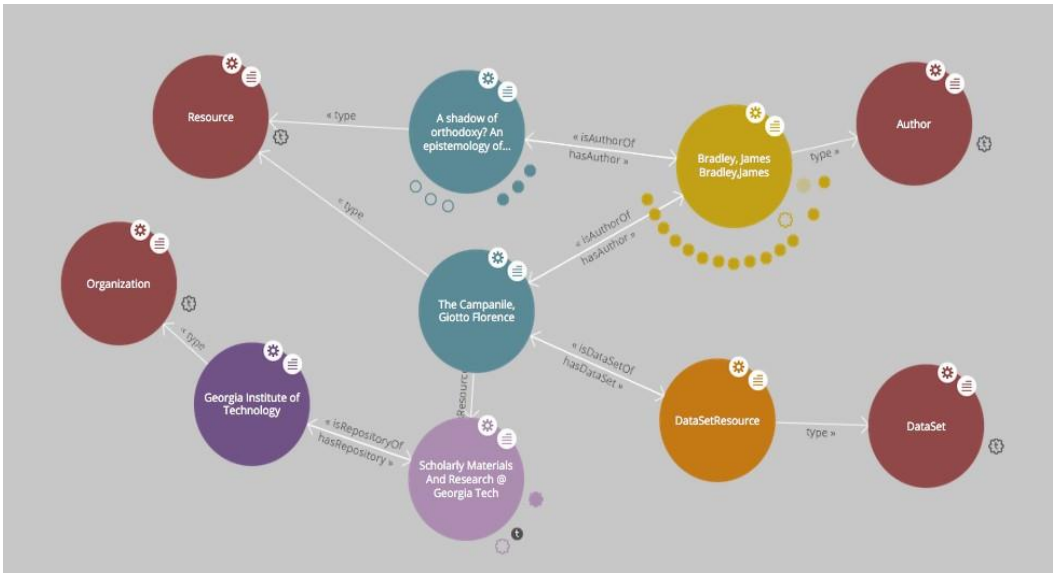

Figure 6 – Representation of the resource with all its metadata with LodLive

2.4 Integration and dependencies

gLibrary 2.0 requires a **MongoDB** (>3.x)[39] server running. It is used to maintain repositories' configurations and also for metadata and data records storage for local repositories. Node.js (> 0.12)[40] is required too.

Being a server service, a tool that ensures it indefinitely runs or restarts automatically is recommended. We have successfully deployed gLibrary both with forever[41] and pm2[42]. For testing purposes, Nodemon[43] is a valid option.

The Parallel Semantic Search Engine service can easily be installed deploying a simple JSR 286 compliant portlet in the Application Server. For this pilot we have tested the Parallel Semantic Search Engine on GlassFish (v3.1.1) as Application Server.

To graphically represent the resource with all its metadata with LodLive[44], the tool has to be installed separately in the application server. Instructions to install and configure LodLive are the following:

```
]$ sh <glassfish_path>/bin/asadmin –u liferayadmin \
        –W <glassfish_path>/domain/liferay/config/local-password deploy LodLiveGraph.war[45]
```

[39] https://www.mongodb.org
[40] https://nodejs.org/
[41] https://github.com/foreverjs/forever
[42] https://github.com/Unitech/pm2
[43] https://github.com/remy/nodemon
[44] https://github.com/dvcama/LodLive

3 Requirements

Cloud computing has already proven it can deliver lower IT costs, reduce infrastructure complexity, enhance flexibility and deliver high-quality new services. This include also the possibility to deal with/exchange dynamic data among computing centres in a user-transparent way, preventing data loss due to localized disasters. In this deliverable we have presented a solution to manage digital repository in cloud-based infrastructure.

3.1 Functional Requirements

The PSSE service:

- Aims to correlate scientific papers to datasets used to produce them and to discover data and documents in an easy way;
- Provides a simple and very intuitive user-friendly interface to allow researchers with limited ICT skills to search keywords in different digital repositories. Results of these queries are presented in table format. ;
- Can be easily deployed on GlassFish Application server and be accessed anytime and everywhere as a Web service;
- Has an harvester process to link Open Access Repositories (OAR) and Data Repositories (DRs) that can be executed both on Grid and Cloud infrastructures.

3.2 Non-functional Requirements

Performance

- Speed and fault tolerance for the parallel SSE have been improved with Java threads to run SPARQL queries in parallel in different Open Access Repositories;
- To avoid users wait too long, a timed-out feature has been included to stop the search if no entries are found in 1 minute

Availability

- Source codes for gLibrary and the SSE services are available on GitHub. INFN will keep documentations and source codes updated;
- Both services will be available 24h, 7*365 for end-users and will be accessed by varying number of users;
- Travis Integration tests are implemented.

[45] https://github.com/osct/LodLiveGraph/blob/master/LodLiveGraph.war

4 Feedback on satisfaction

A number of humanities related searches were attempted (linguistics, history) and the search engine performed as expected on each occasion. The software used, the features and the architecture are outlined and exhaustive references to the software used are reported. The document explains in a clear way the background framework of the DARIAH competence centre and the two use cases (mini-projects) developed within it.

5 Future plans

To raise the awareness of A&H researchers about the necessity of digital research, the two services will be presented during workshops, webinars and/or training courses that will be organized by the DARIAH CC project.

This will contribute to advertise the project results and, based on the feedback received from the community, pave the way to drive the further development of these services and/or identify new areas of collaborations.

Future developments include the possibility to:

- Enrich the information accessed through the Parallel Semantic Search Engine including some general-purpose and domain-specific repositories (e.g. GENONAMES[46]) and dictionaries (e.g GERMANET[47] andISLEX[48]);
- Extend the ontology based on Protègè to include DARIAH-specific knowledge;
- Discuss about possible integration with other high-level tools developed in the context of the DARIAH CC project;
- Include other DARIAH-specific tools.

[46] www.genonames
[47] www.sfs.uni-tuebingen.de/GermaNet/
[48] www.islex.is/

Appendix I. gLibrary 2.0 API Docs

Server endpoint

```
http://glibrary.ct.infn.it:3500
```

Authentication

Before sending any request, users must be authenticated. Currently, the authentication is based on username/password couple. This will return a **session token id** that needs to be used with any of the following requests. There are two options to send the **access_token**:

- via a query parameter:

```
curl -X GET http://glibrary.ct.infn.it:3500/v2/repos?access_token=6Nb2ti5QEXIoDBS5FQGWIz4poRFiBCMMYJbYXSGHWuulOuy0GTEuGx2VCEVvbpBK
```

- via HTTP headers:

```
ACCESS_TOKEN=6Nb2ti5QEXIoDBS5FQGWIz4poRFiBCMMYJbYXSGHWuulOuy0GTEuGx2VCEVvbpBK

curl -X GET -H "Authorization: $ACCESS_TOKEN" \
http://glibrary.ct.infn.it:3500/v2/repos
```

Login

To obtain a session id, you need to pass a valid *username* and *password* to the following endpoint:

```
POST /v2/users/login HTTP/1.1
```

```
{
 "username":"admin",
 "password":"opensesame2015"
}
```

Alternatively you can use the *email* address instead of the *username*.

User creation

New users are created issuing requests to the following endpoint:

```
POST /v2/repos/users HTTP/1.1
```

The mandatory parameters are:

- **username**
- **email**
- **password**

Please notice that the created user has no access to any repository yet. The admin user needs to assign the created user to one or more repositories and/or collections, by properly setting the ACLs (see below).

Authorization

Currently, gLibrary allows the setting of separate permissions to repositories, collections and items for each user. The default permission set to a newly created user is *NO ACCESS*. It is admin's responsibility to properly set the ACLs per each user. Currently an instance of gLibrary server has just one super-admin (the *admin* user), but in future releases will allow to define more administrators per repository.

ACLs

To set ACLs, the super admin can issue requests to two separate endpoints:

```
POST /v2/repos/<repo_name>/_acls http/1.1
```

and/or

```
POST /v2/repos/<repo_name>/<collection_name>/_acls http/1.1
```

The body of each request has the following attributes:

attribute	Description
username	the username of the user to which we are adding permissions to
permissions	valid options are "R" and "RW"
items_permissions	(for collections only) valid options are "R" and "RW"

permissions refers to repository or collection permission, according to where the request is issued:

- Repository:
  - "R" grants a user the capability of listing its content (i.e. list of collections)
  - "RW" grants a user the capability of creating (or importing) new collections or deleting them
- Collection:
  - "R" grants a user the capabilities to list the collection's content (list of items)
  - "RW" grants a user the capabilities of creating, updating, deleting the collection's items

items_permissions is valid only for collections' ACL and refers to:

- "R" grants a user the capability to download items' replicas
- "RW" grants a user the capability to create, update and upload replicas

Repositories

A gLibrary server can host one or more **repositories**. A repository should be created before creating new **collections** or importing existing db tables or NoSQL collections as gLibrary collections.

A repository has a *name*, a *path* that represents the access point in the API path, and optionally a *coll_db* (*TODO*: rename as *default_collection_db*). If the default DB is defined at the moment of creation, this will be the default backend DB for all the collections created or imported of the given repository. However, this can be overridden per each collection, if new DB info is provided when the collection is created

List of all the repositories hosted on the server

```
GET /v2/repos/ HTTP/1.1
```

Returns a list of all the repositories managed by the given gLibrary server. Each repository has the following properties:

name	Description
name	Repository name
path	Direct endpoint of the given repository
collection_db	Default database where collection data should be stored. Can be overriden per collection
host	FQDN of the default collection DB
port	port number of the default collection DB
username	username of the default collection DB
password	password of the default collection DB
database	name of the database to use for the default collection DB
type	type of the default collection db (mysql, postgresql, mongodb)

Example:

```
{
    "name": "infn",
    "path": "http://glibrary.ct.infn.it:5000/v2/infn",
    "coll_db": {
        "host": "giular.trigrid.it",
        "port": 3306,
        "username": "root",
        "password": "**********",
        "database": "test",
        "type": "mysql"
    }
}
```

Each repository can have a *collection_db* where collections data will be stored. If no *collection_db* is specified, the repository will use the local non-relational mongoDB that comes with gLibrary. Each repository's collection can override the *collection_db*.

Create a new repository

```
POST /v2/repos/ HTTP/1.1
```

Create a new repository. A default *collection_db* can be specified. It will store all the collections in case no *collection_db* parameter is specified during collection creation. This property is optional. If missing, it will use the local MongoDB server.

Parameters

name	type	Description
name	string	Name of the repository (will be the API path)
collection_db	object	(Optional) Default database where collection data should be stored. Can be overridden per collection
host	string	FQDN of the default collection DB
port	number	port number of the default collection DB
username	string	username of the default collection DB
password	string	password of the default collection DB
database	string	name of the database to use for the default collection DB
type	string	type of the default collection db (mysql, postgresql, mongodb)
default_storage	object	(Optional) specifies the default storage for replicas
baseURL	string	it's full path of Swift Container or Grid SURL for replica uploads
type	string	"swift" or "grid" storage

Note: *name* is a lowercase string. Numbers and underscores are allowed. No special characters are permitted

Example:

```
POST /v2/repos/ HTTP/1.1
Content-Type: application/json

{
    "name": "infn",
    "collection_db": {
        "host": "glibrary.ct.infn.it",
        "port": 5432,
        "username": "infn_admin",
        "password": "******",
        "database": "infn_db",
        "type": "postgresql"
    },
    "default_storage": {
        "baseURL": "http://stack-server-01.ct.infn.it:8080/v2/AUTH_51b2f4e
508144fa5b0c28f02b1618bfd/gridcore",
        "type": "swift"
    }
}
```

Be sure to set *Content-Type* to *application/json* in the *Request Headers*.

Collections

Each repository contains one or more collections. Collections are abstractions over relational database tables or non-relational database "collections", exposing their records over REST API and JSON format. The available API allows the repository administrator to create new collection, specifying a schema in the case of relational collection, or importing existing tables/NoSQL collections. If not specified, collections will be created/imported from the default *coll_db* (*TODO*: *default_collection_db*) of the containing repository. Otherwise, each collection can retrieve data from local or remote database, overriding the default repository value, using the *coll_db* (*TODO*: *collection_db*) property.

Create a new collection

```
POST /v2/repos/<repo_name>/ HTTP/1.1
```

Parameters

name	type	description
name	string	Name of collection
schema	object	(Optional for non-relational DB) define the schema of the new collection
collection_db	string	(Optional) Default database where collection data should be stored. Can be overridden per collection
host	string	FQDN of the default collection DB
port	number	port number of the default collection DB
username	string	username of the default collection DB
password	string	password of the default collection DB
database	string	name of the database to use for the default collection DB
type	string	type of the default collection db (mysql, postgresql, mongodb)

Schema is a JSON object listing the name of the attributes and their types in case we want a non-relational collection. Each property represents the name of an attribute and the value is another object with the following keys:

name	description
type	type of the attribute's value. Example of allowed types are: string, number, 'boolean', 'date'
required	whether a value for the property is required
default	default value for the property
id	whether the property is a unique identifier. Default is false

For a full list of the supported type, please refer to https://docs.strongloop.com/display/public/LB/LoopBack+types and https://docs.strongloop.com/display/public/LB/Model+definition+JSON+file#ModeldefinitionJSONfile-Generalpropertyproperties.

Example (creation of a new collection on a relational db):

```
POST /v2/repos/infn/ HTTP/1.1
Content-Type: application/json

{
    "name": "articles",
    "schema": {
        "title": {"type": "string", "required": true},
        "year": "integer",
        "authors": "array"
    }
}
```

The previous request will create a collection named *articles* into the *infn* repository. The collection data will be stored into the default *coll_db* specified for the *infn* repository (that according to the previous example is a postgreSQL db named *infn_db*)

Import data from an existing relational database

If you want to create a collection that maps an existing db table, two additional properties are available:

name	description
import	it should set to *true*
tablename	name of the database table of the database to be imported

Example (creation of a new collection with data coming from an existing relational db):

```
POST /v2/repos/infn/ HTTP/1.1
Content-Type: application/json

{
    "name": "old_articles",
    "import": "true",
    "tablename": "pubs",
    "collection_db": {
        "host": "somehost.ct.infn.it",
        "port": 3306,
        "username": "dbadmin",
        "password": "******",
        "database": "test_daily",
        "type": "mysql"
    }}
```

The previous request will create the collection *old_articles* import data from an existing database, named *test_daily* and providing access to its table named *pubs*.

List all the collections of a repository

```
GET /v2/repos/<repo_name>/ HTTP/1.1
```

This API will return a JSON array with all the collections of *<repo_name>*. Each collection will have a *schema* attribute, describing the schema of the underlying DB table. If the *schema* attribute is *null* it means the collection has been imported and it inherits the schema of the underlying DB table. An additional API is available to retrieve the schema of a given collection (see next API).

Example

```
GET /v2/repos/sports HTTP/1.1
```


```
[
    {
        "id": "560a60987ddaee89366556d2",
        "name": "football",
        "path": "/sports/football",
        "location": "football",
        "coll_db": null,
        "import": "false",
        "schema": null
    },
    {
        "id": "560a60987ddaee89366556d3",
        "name": "windsurf",
        "path": "/sports/windsurf",
        "location": "windsurf",
        "coll_db": null,
        "import": "false",
        "schema": {
            "rider": {
                "type": "string",
                "required": true
            },
            "nationality": {
                "type": "string",
                "required": false
            },
            "teamid": {
                "type": "number",
                "required": false
            }
        }
    }
]
```


The *sports* repository has two collections *football* and *windsurf*. The first one is stored on the default *coll_db* repository DB and its schema-less, while the second one has a predefined *schema*.

Retrieve the schema of a collection

```
GET /v2/repos/<repo_name>/<collection_name>/_schema HTTP/1.1
```

If the given *collection_name* is hosted in a relation database table, this API will return a JSON object with the schema of the underlying table.

Example

```
GET /v2/repos/comics/dylandog/_schema HTTP/1.1
```


```
{
    "id": {
        "required": true,
        "length": null,
        "precision": 10,
        "scale": 0,
        "id": 1,
        "mysql": {
            "columnName": "id",
            "dataType": "int",
            "dataLength": null,
            "dataPrecision": 10,
            "dataScale": 0,
            "nullable": "N"
        }
    },
    "fragebogenId": {
        "required": true,
        "length": null,
        "precision": 10,
        "scale": 0,
        "mysql": {
            "columnName": "fragebogen_id",
            "dataType": "int",
            "dataLength": null,
            "dataPrecision": 10,
            "dataScale": 0,
            "nullable": "N"
        }
    },
    "nummer": {
        "required": true,
        "length": 256,
        "precision": null,
        "scale": null,
        "mysql": {
            "columnName": "nummer",
            "dataType": "varchar",
            "dataLength": 256,
            "dataPrecision": null,
            "dataScale": null,
            "nullable": "N"
```

```
        }
    }
}
```

Delete a collection

```
DELETE /v2/repos/<repo_name>/<collection_name>  HTTP/1.1
```

This API will delete the given *collection_name* from *repo_name*. Actual data on the backend table should not be deleted. It's a sort of *unlinking*, so that the db table/nosql collection will not be accessible anymore from the gLibrary REST API.

Items (previously entries)

Items represent the content of a given collection. If a collection is hosted in a relational database, each item is a table record, while if it's non-relational it's the document/object of the NoSQL collection. Items can be listed and queried via the filtering system, created/added, updated and deleted, using the REST API provided by gLibrary.

Item creation

```
POST /v2/repos/<repo_name>/<collection_name> HTTP/1.1
```

This API adds a new item into the given *collection_name*. Item content have to be provided as a JSON object. In case of the relational collection it should conform to the collection schema. In the case of attributes that have no corresponding column table, their values will be ignored silently. If the API will be successful a new record or document will be added to the underlying table or NoSQL collection.

Example

```
POST /v2/repos/infn/articles HTTP/1.1

{
    "title": "e-Infrastructures for Cultural Heritage Applications",
    "year": 2010,
    "authors": [ "A. Calanducci", "G. Foti", "R. Barbera" ]
}
```

Item listing

```
GET /v2/repos/<repo_name>/<collection_name>/ HTTP/1.1
```

Retrieve the items inside the *collection_name* as a JSON array of objects. Each object is a record of the underlying table (in case of relational DB) or document (in case of NoSQL collection). By default the first 50 items are returned. See below the description of filtering system in the query section to change this behavior.

Example

```
GET /v2/repos/gridcore/tracciati     HTTP/1.1
```

Item detail

```
GET /v2/repos/<repo_name>/<collection_name>/<item_id> HTTP/1.1
```

Retrieve the detail of an item with a *given_id*. It will return a JSON object with the attributes mapping the schema of the given *collection_name*.

Example

```
GET /v2/repos/infn/articles/22
```

Item deletion

```
DELETE  /v2/repos/<repo_name>/<collection_name>/<item_id> HTTP/1.1
```

Delete the given *item_id* of the the collection *collection_name*. Delete will be successful only if the given item has no replica. You can force the deletion of item with replicas setting:

```
{
    "force": true
}
```

in the request body.

Item update

```
PUT /v2/repos/<repo_name>/<collection_name>/<item_id> HTTP/1.1
```

Update one of more attributes of the given *item_id*. The request body has to contain a JSON object with the attribute-value pair to be updated with the new values.

Queries with filters

```
GET  /v2/repos/<repo_name>/<collection_name>?filter[<filterType>]=<spec>&fi
lter[...]=<spec>... HTTP/1.1
```

where *filterType* is one of the following:

- *where*
- *include*
- *order*
- *limit*
- *skip*
- *fields*

and *spec* is the specification of the used filter.

Additional info on the full query syntax can be found here

Replicas

Each item can have one or more attachments, generally the same file stored in different locations, such as Cloud storage servers (Swift based) or Grid Storage Elements (DPM based). So we call them also replicas.

Replica creation

```
POST /v2/repos/<repo_name>/<collection_name>/<item_id>/_replicas/ HTTP/1.1
```

name	description
uri	(optional) provides the full storage path of where the replica will be saved
type	(optional) specifies the type of storage backend. Currently "swift" or "grid"
filename	The filename of the given replica

The first two parameters (*uri* and *type*) are optionals if a *default_storage* attribute has been set for the given repository. If not, they need to be specified; otherwise the request to the API will fail.

Please note that this API will just create a replica entry for the item, but no actual file will be uploaded from the client. Once the replica has been created you need to use the **Upload** API to transfer the actual file payload.

Retrieve all the replicas of the given *item_id*

```
GET /v2/repos/<repo_name>/<collection_name>/<item_id>/_replicas/ HTTP/1.1
```

Download a given replica

```
GET /v2/repos/<repo_name>/<collection_name>/<item_id>/_replicas/<rep_id> HTTP/1.1
```

Upload a replica

Upload the file payload to the destination storage. This requires two subsequent API request.

First, ask for the destination endpoint for the upload with:

```
PUT /v2/repos/<repo_name>/<collection_name>/<item_id>/_replicas/<rep_id> H
TTP/1.1
```

This will return a **temporaryURL** valid a few seconds (example):

```
{
  "uploadURI": "http://stack-server-01.ct.infn.it:8080/v2/AUTH_51b2f4e5081
44fa5b0c28f02b1618bfd/gridcore/ananas.jpg?temp_url_sig=6cd7dbdc2f9e429a1b8
9689dc4e77f1d2aadbfc8&temp_url_expires=1449481594"
}
```

Then use the URL returned by the previous API to upload the actual file, using the PUT verb again (example):

```
PUT http://stack-server-01.ct.infn.it:8080/v2/AUTH_51b2f4e508144fa5b0c28f0
2b1618bfd/gridcore/ananas.jpg?temp_url_sig=6cd7dbdc2f9e429a1b89689dc4e77f1
d2aadbfc8&temp_url_expires=1449481594 HTTP/1.1
```

It will return a 201 status code, if the upload will complete successfully

Delete a replica

```
DELETE /v2/repos/<repo_name>/<collection_name>/<item_id>/_replicas/<rep_id
> HTTP/1.1
```

Relations

One to many relations can be created between collections of the same repository, properly setting a foreign key.

To set the relation among two collections, issue the following request to the collection in the "one" side of the one-to-many relation:

```
POST /v2/repos/<repo_name>/<collection_name>/_relation HTTP/1.1
```

The body of the request needs to provide two attributes:

name	description
relatedCollection	the "many" side of the one-to-many relation
fk	the *foreign key* of *relatedCollection* that match the *id* of <*collection_name*>

In practice, you should set the *fk* in such a way *`collection_name.id`* == *`relatedCollection.fk`*

Retrieve related items

```
GET /v2/repos/<repo_name>/<collection_name>/<item_id>/<related_collection>
```

Retrieve all the items from *`related_collection`* of the given *`item_id`*.

Contributors

Antonio Calanducci (antonio.calanducci@ct.infn.it)

Antonio Di Mariano (antonio.dimariano@gmail.com)