H2020-INSO-2014
INSO-1-2014 ICT-Enabled open government
YDS [645886] “Your Data Stories”

D3.10 Open Data Repository v2.0

Project Reference No	645886 — YDS — H2020-INSO-2014-2015/H2020-INSO-2014
Deliverable	D3.10 Open Data Repository v2.0
Workpackage	WP3: Data Layer
Nature	DEM
Dissemination Level	PU
Date	29/12/2016
Status	Final v1.0
Editor(s)	Aad Versteden, Uroš Milošević, Erika Pauwels (TF)
Contributor(s)	George Petasis (NCSR-D)
Reviewer(s)	Michalis Vafopoulos (NCSR-D)
Document description	This deliverable will constitute the YDS open data repository prototype. This repository will offer harvested and possibly interlinked data to other YDS components efficiently and effectively.

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
V0.1	27/09/2016	Table of Contents	TF
V0.3	05/11/2016	Triple store and graphs	TF
V0.4	25/11/2016	Mu-semte-ch improvements	TF, NCSR-D
V0.6	02/12/2016	JSON-LD & Apache Solr	NCSR-D
V0.7	07/12/2016	API Documentation	TF, NCSR-D
V0.8	16/12/2016	Ready for peer review	TF
V0.9	22/12/2016	Peer reviewed	NCSRD-D
V1.0	23/12/2016	Final	TF
V1.0	29/12/2016	Final version for submission to EC	ATC

Executive Summary

The YDS Open Data Repository hosts the contents for widely varying views on a collection of large interlinked datasets. We support this by offering a platform with both standardized and custom data-offering services. The extended variant of the open data repository supports SPARQL [1], JSON API[2] and JSON-LD[13] based calls. Moreover, its natural language search capabilities have been extended with the inclusion of an Apache Solr [14] instance.

The YDS project can offer very different views on the same ground truth. The information which is available through the unified data model is shown to users in new and exciting ways. The goal is to offer insights based on innovative visualizations. This provides challenges in terms of performance and display of the necessary content for the Open Data Repository.

The ever-increasing size of the data calls for a more careful data management approach, which is why special attention is given to dataset description, virtual data aggregation and scheduled backups.

The core of the architecture of the Open Data Repository is based on mu.semte.ch [3, 4]. This project provides a microservices based architecture for offering views on linked data. Central in this architecture is the triplestore, which contains all the information on the platform. Many microservices can offer support for updating and viewing the data in the triplestore. These microservices can be developed by the repository developers, or independently by component developers.

JSON API views on the YDS data model are offered by use of the mu-cl-resources [5] project. This microservice acts as the glue between the contents of the JSON responses, and the linked data model. SPARQL access is offered by Virtuoso [6]. A single endpoint for all microservices is offered by the mu-dispatcher [10], which is configured accordingly. The JSON APIs are offered by an API which adheres to the JSON API specification. This will limit internal discussions and help developers find tooling for their development environment. Deployment is made easy by using Docker [8]. The whole system is orchestrated using Docker Compose [9].

Additionally, JSON-LD compliant responses are provided through an additional, more descriptive Linked Data API. The API provides the resource contextual/interpretation metadata, and makes JSON data interoperate at Web-scale. Full developer documentation is provided for both JSON-based APIs.

To improve the natural language search performance and provide advanced full-text search capabilities, all natural language fields are indexed in Apache Lucene, the underlying engine in Apache's standalone enterprise search platform, Solr.

Table of Contents

1 INTRODUCTION 6
1.1 PURPOSE AND SCOPE 6
1.2 APPROACH FOR WORK PACKAGE AND RELATION TO OTHER WORK PACKAGES AND DELIVERABLES 6
1.3 UPDATES WITH RESPECT TO D3.9 7
1.4 METHODOLOGY AND STRUCTURE OF THE DELIVERABLE 7
2 PROBLEM SCOPE 8
3 EXTENDED ARCHITECTURE 9
3.1 TRIPLE STORE 9
3.1.1 Data catalog and interoperability 10
3.1.2 Graphs 10
3.1.3 Backups 12
3.2 MU-SEMTE-CH 12
3.2.1 Layers 12
3.2.1.1 Identifier 13
3.2.1.2 Dispatcher 13
3.2.1.3 Microservices 13
3.2.2 Caching 14
3.2.2.1 Mu-cache 14
3.2.3 Implementation of the model and extensions 14
3.3 EXTENSIONS 15
3.3.1 JSON-LD API 15
3.3.2 Apache Solr 15
4 AVAILABLE APIS 17
4.1 JSON API 17
4.2 JSON-LD API 19
5 DEVELOPER DOCUMENTATION 21
6 FUTURE VISION 23
7 CONCLUSION 24
8 REFERENCES 25

List of Figures

Figure 1: The extended general architecture ... 9
Figure 2: Mu-semte-ch architecture ... 12
Figure 3: JSON API Documentation ... 21
Figure 4: JSON-LD API interactive documentation on SwaggerHub ... 22

List of Terms and Abbreviations

Abbreviation	Definition
SPARQL	SPARQL Protocol and RDF Query Language
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
API	Application Programming Interface
REST	Representational State Transfer

1 Introduction

1.1 Purpose and Scope

The goal of this task is to develop the main repository where all data will be stored, after they have been collected, cleaned, validated, aligned, and cross-linked in Task 3.3. In its final version, this repository will fully implement the data model that will be defined in Task 3.2, and will make the data available to all components of the YDS platform, as well as YDS users, through suitable semantic services. This Open Data Repository will be the central component for managing the data that will be provided by YDS, not only to the rest of the platform layers, but also to all applications that operate on top of the YDS platform, constituting the repository an essential element of the YDS platform offerings. The repository will offer search services, including REST and SPARQL endpoints, delivering results in multiple formats to satisfy the needs of the YDS pilot and third-party applications.

With a considerable amount of data in place in year two of the project, as a direct result of modeling (Task 3.2) and harvesting (Task 3.3) efforts in WP3, the second version of the Open Data Repository represents a significant update over its predecessor bringing numerous features and improvements. New channels for searching and retrieving the data have been delivered that YDS platform components and developers can safely rely on. More details with respect to the updates over D3.9 Open Data Repository 1.0 are provided in Section 1.3.

1.2 Approach for Work Package and Relation to other Work Packages and Deliverables

The primary focus of the third Work Package is data collection, management, storage, and provision to other components of YDS through effective programming services and interfaces. The relevant technologies can be indexed in four main layers:

1. Harvesters, which acquire data from open data sources and the social Web.
2. Validators, assessing the quality of harvested data, cleaning data, and validating them according to the data model requirements.
3. Ontology population and alignment, where diverse data sources are aligned, cross-linked, and transformed to use a common vocabulary, ultimately becoming part of a unifying data model.
4. Data storage and querying, where data are accumulated in a data store, which provides retrieval and management through suitable semantic interfaces (such as SPARQL queries).

In particular, the third layer provides a common language across all the four layers by enabling the orchestration of diverse specifications from heterogeneous systems and methodologies. Actually, it serves as a detailed and interactive reference of the modeling choices that have been made for all relevant data sources.

1.3 Updates with respect to D3.9

In the second year of the project, we focused on establishing a reliable and performant data layer, while easing the data exploration and retrieval for third-party developers and the YDS consortium alike.

At the time of writing this deliverable the production-ready YourDataStories database stores over 20 million triples in 18 different RDF graphs. The ever-increasing size of the data calls for a more careful data management approach, which is why special attention is given to dataset description, virtual data aggregation and scheduled backups. The high-level approach is described in this report and will be further detailed in D2.8 Data Management Plan V2.0.

Moreover, we present extensions of the overall architecture in terms of new services and the improvements to existing ones (including both functional and performance improvements, as foreseen by D3.9).

The JSON API compliant *mu-cl-resources* microservice has been extended to support more of the JSON API specification, including features such as sorting, inclusion of related resources, and sparse fieldsets. Additionally, we provide better support for Linked Data specific features, such as resource URIs, language tags and geometries.

Furthermore, developers with Linked Data expertise can now also traverse the YDS graphs via a JSON-LD (JavaScript Object Notation for Linked Data) API, as an additional channel.

Also, natural language search capabilities and performance are improved through the addition of an Apache Solr powered search layer.

Finally, complete developer documentation is provided for both *mu-cl-resources* (JSON API) and JSON-LD APIs.

1.4 Methodology and Structure of the Deliverable

This deliverable builds on top of D3.9 and, hence, shares some of its structure. The topics in the deliverable can be considered to be sectioned in chronological order. The first portions of the project should be constructed before the latter ones can be used. The topics by which the deliverable was split, indicate different uses of the platform. Given some particular interest, it should be clear which topics will be of interest to the reader.

The first section aims to build a shared understanding of the technical problems in the problem domain. We don't make claim to understand all the problems in this domain, but rather seek to find a general direction of issues. With these constraints in mind, we focus on the general architecture of a platform that could solve these needs. We follow up by discussing in what way the platform needs to be constructed in order to support the model of the YDS platform. The APIs, which are the result of this, as well as the improvements, are then discussed. Lastly, we present a vision on how the repository could be extended even further in order to cope with the future needs of the YDS project.

2 Problem scope

The YDS platform will cater for widely varying views on showing publicly available content. The platform also needs to be comparatively easy to install, whilst still offering a way to scale up in the future. The Open Data Repository receives contents from T3.3, but needs to cope with the structure and offer views on the data. In this section, we try to derive the problems which the Open Data Repository will need to cope with. We don't make claim to cover the whole problem space, but aim to offer support for at least these challenges.

The users of the data offered by the YDS platform might be linked data experts, but we aim to support other users too. Some of them may have limited understanding of the linked data model. The platform should also cater for web developers who mainly want to gain and share insights. The linked data model allows us to easily integrate the data, but it should not be a requirement for those using the platform. We also don't want to scare linked data experts away. Offering an endpoint which talks in common formats should also be offered. Given the web-based nature of the platform, we mainly foresee JSON to play an important role here.

Although the repository at this stage does not need to scale to very large amounts of data, the architecture should not limit such use. It should be easy to scale up the platform, or it should be possibly to evolve the platform so scaling is possible. Ideally, this should not make the platform too complex to install either. Especially for the non-scaled repository, it should not require multiple man-days to install.

Moreover, different YDS components may require different views on the same data. We should try to minimize the footprint of the APIs we offer. Maintaining a large code-base can be costly. However, with the widely varying components being offered, we should foresee that at least some of these components will require some custom views on the data. It is tempting to only foresee altering views on the data, but we should also support server-side computations and caching for some components. A way to install and run plugins on the repository would help in supporting the data output for new components.

Finally, as one of the most common data exploration use cases is still natural language search, there is a need for an enterprise search layer which would overcome the shortcomings of triple store (i.e. SPARQL) search capabilities.

3 Extended architecture

The core of the overall architecture of the Open Data Repository is *mu-semte-ch*, the user-facing microservice architecture, relying on an Open Link Virtuoso as a triple store. This is further extended with the inclusion of JSON-LD as a more descriptive Linked Data API, able to also provide the resource contextual/interpretation metadata, and Apache Solr as a more potent search solution (Figure 1).

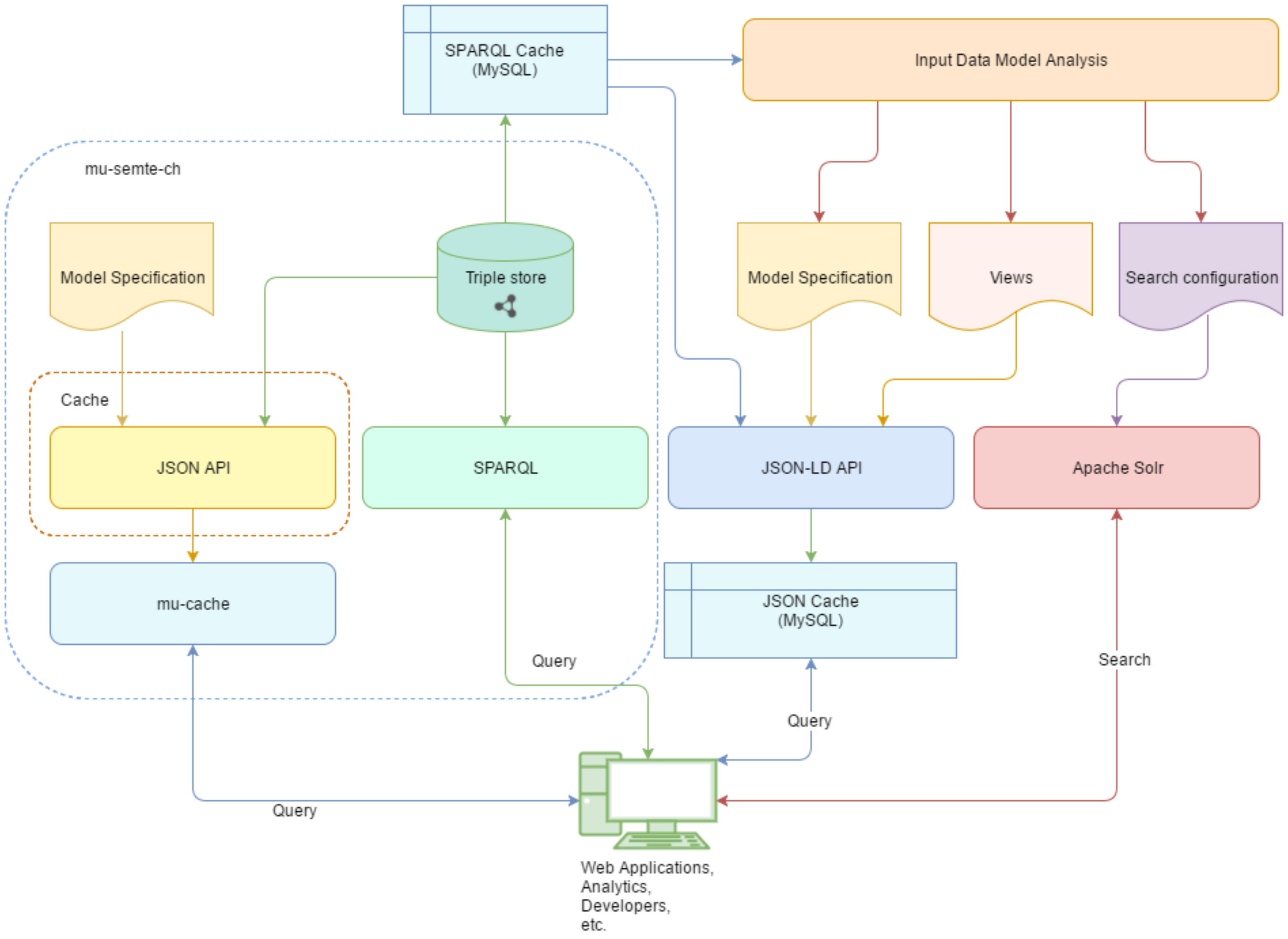


Figure 1: The extended general architecture

3.1 Triple store

The YDS triple store is an OpenLink Virtuoso[1] instance, a middleware and database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, web application server and file server functionality in a single system. Even though OpenLink labels Virtuoso as a "universal server", the YDS Open Data Repository relies only on the set functionalities it excels at, i.e. its graph storage and querying capabilities. Virtuoso provides the most powerful, native Linked Data API out of the box – SPARQL. The YDS SPARQL 1.1 compliant endpoint is publicly

[1] https://virtuoso.openlinksw.com

available.[2] For security reasons, all SPARUL features have been disabled, and are accessible only via iSQL.

3.1.1 Data catalog and interoperability

The description of every harvested and publicly dataset is provided in terms of a fully compliant DCAT-AP[3] entry contained in a publicly available DCAT-AP catalog which can be retrieved via any of the available APIs. This means that every dataset has a corresponding natural language description, publisher, temporal, thematic, access and, most importantly, licensing information.

Moreover, DCAT-AP ensures interoperability and catalog harvestability by other DCAT-AP compliant data catalogs in Europe, such as the European Data Portal[4].

3.1.2 Graphs

With over 20 million triples in 18 different RDF graphs and constantly growing amount of data being served, the fragmentation, combined with the complexity of the data can make exploration difficult. To truly exploit the linked nature of the data, the data is aggregated to form a virtual graph group, as shown in the table below.

Table 1: OpenLink Virtuoso Graphs

	Graph URI	Notes
Common		
EuroVoc	http://yourdatastories.eu/taxonomies/eurovoc	The EuroVoc SKOS taxonomy. Contained in a separate graph because it is considerably larger than the others.
Aggregates		
Everything below, physically integrated	http://mu.semte.ch/application	Physical integration is (currently) needed by the JSON API. For backend operations (platform & components), the virtual graph (below) is recommended.
Everything below, virtual graph	http://yourdatastories.eu/virtual	Virtuoso virtual graph group.
Individual		
Common		
Taxonomies / Code lists	http://yourdatastories.eu/taxonomies	All SKOS taxonomies & code lists
T-Box	http://yourdatastories.eu/ontology	Full ontology specification, concept labels & comments.
Dataset catalog	http://yourdatastories.eu/catalog	Contains all DCAT-AP dataset descriptions

[2] http://yds.iit.demokritos.gr:8890/sparql
[3] https://joinup.ec.europa.eu/asset/dcat_application_profile/description
[4] https://www.europeandataportal.eu

Countries	http://yourdatastories.eu/countries	Supporting information about countries, enriched with data from DBpedia and the FAO Geopolitical Ontology
World Factbook	http://yourdatastories.eu/WorldFactbook	Supporting information about countries (natural resources, industries, agricultural products) provided by the CIA
World Development Indicators	http://yourdatastories.eu/WDI	Supporting information about countries provided by the World Bank
Open Corporates	http://yourdatastories.eu/OpenCorporates	Supporting data about (mostly pilot 2) organizations provided by Open Corporates
Development regions	http://yourdatastories.eu/regions	IATI Development regions
Pilot 1		
Greek Transparency Portal (Diavgeia)	http://yourdatastories.eu/Diavgeia	Data provided by Greek Transparency Portal (Diavgeia)
National Strategic Reference Framework (NSRF)	http://yourdatastories.eu/NSRFUpdate	Data provided by National Strategic Reference Framework (NSRF)
Pilot 2		
Official Development Assistance (From: Netherlands)	http://yourdatastories.eu/ODA/NL	Data provided by IATI. Also, includes transaction data.
Official Development Assistance (To: Zimbabwe)	http://yourdatastories.eu/ODA/ZW	Data provided by IATI
International Trade (Netherlands)	http://yourdatastories.eu/trade/NL	Data provided by MIT (UN Comtrade Database)
International Trade (Zimbabwe)	http://yourdatastories.eu/trade/ZW	Data provided by MIT (UN Comtrade Database)
International Trade Agents	http://yourdatastories.eu/tradeagents	Supporting data about International Trade, linking trade agents to corresponding countries (used to avoid repetition in individual trade graphs)
Pilot 3		
TED	http://yourdatastories.eu/TED	Tenders Electronic Daily as Linked Data

3.1.3 Backups

Even though the data on the development server is never made public, both the development and the production server have failsafe mechanisms in place for the data and the associated harvesting processes. The data on both servers is backed up once a week on local disk, i.e. every Tuesday, at 4 AM (i.e. outside peak hours). Moreover, the harvesters are backed up on GitHub, in a dedicated repository[5], ensuring fast recovery even in case of loss of data. The harvesters are described in more detail in D3.7 Data Harvesters v2.0.

3.2 Mu-semte-ch

As previously mentioned, the *mu-semte-ch* part of the architecture of the Open Data Repository is based on user-facing microservices which store their data in the above-described, shared triplestore. The architecture has grown together with the mu.semte.ch project [3, 4]. Each of the microservices are said to be user-facing, in that they offer a service aimed directly at the end-users. The effort of installing all these microservices is minimized by bundling them in a Docker Compose [9] configuration. This allows the whole system to be downloaded and installed through a single command.

3.2.1 Layers

A request for the Open Data Repository passes through 4 layers (Figure 2). The first layer, *mu-identifier* [11], identifies the device that is accessing the service. The second layer, *mu-dispatcher* [10], ensures the request is dispatched to the right microservice. The third layer is one of the microservices. It communicates with the triplestore to return a response to the user. Except for a possible cookie set to identify the current user, the response is left untouched. This makes reasoning on the services easy. Each of these phases will be described in the following sections.

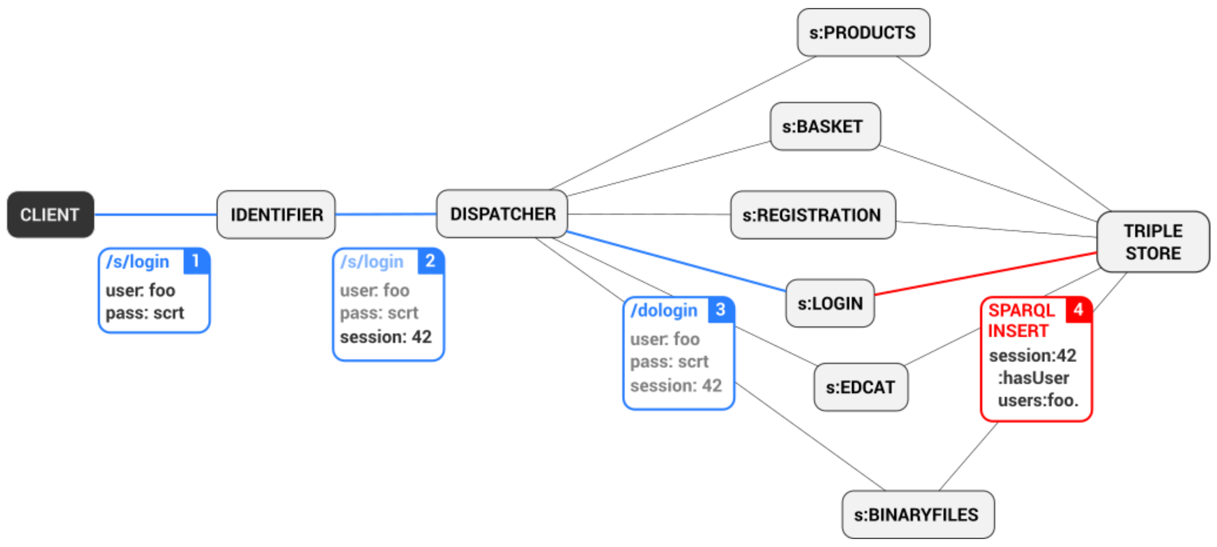


Figure 2: Mu-semte-ch architecture

[5] https://github.com/YourDataStories/harvesters

3.2.1.1 Identifier

The identifier layer is a trivial microservice which sets a cookie in the user's browser. This cookie is used to identify the current device which is accessing the service. When a user logs in through a login service, this service can store in the database that a user has logged in on the system with that device. Other services can then use that information if they need to support the currently logged in user. Logging out of the system would consist of removing the triple which connects the device identifier to the session. The identifier identifies a device and augments the request with a custom HTTP header containing an identification URI for the current device.

3.2.1.2 Dispatcher

The dispatcher contains a configuration to select the right microservice to handle a specific request. A specific microservice could be hosted on a variety of URLs. Many microservices will not care about the path on which they are hosted. A service which returns the total amount of currently logged in users would not need to adapt whether it is hosted on /stats/logged-in-users or /info/users/logged-in-count. The dispatcher decides which service is available on which path and forwards an incoming request to the correct microservice. Note, however, that some microservices may care about the location on which they are hosted.

3.2.1.3 Microservices

The microservices contain the specific implementations for the YDS components. One specific microservice is *mu-cl-resources*, which maps the contents of the triplestore to a JSON API [2] compatible REST API. The microservice receives the request and uses a connection to the triplestore to show and/or alter the contents of the triplestore. The response generated by the microservice is sent back to the end-user. The microservices can cooperate without much synchronization, as they all work on the same standardized data-model. The semantic model in the triplestore is therefore key for the correct operation of the architecture.

This whole setup of microservices is maintained through the Docker ecosystem. A Docker container can be considered to be a lightweight virtual machine, ideally hosting a single application. In our case, each of the microservices will be run in a single Docker container. The whole system is packaged in a *docker-compose* file. This file orchestrates a set of Docker containers and how they should work together. You can start the orchestration by running a single command, which will download and start all necessary Docker containers.

The microservices in the mu.semte.ch project can all be scaled up, aside from the database (in its open source variant). Much of the architecture is, therefore, rather attractive to use at scale. Replication and load balancing would work for most of the components.

Our approach of using microservices makes it easy to implement a new microservice for a specific need. If there is a component in the YDS platform which needs a different approach than the standardized JSON API based responses, it can be implemented in a microservice. In contrast to other approaches, much of the complexities of working with microservices are hidden due to the use of the shared model in the triplestore.

3.2.2 Caching

Two layers of caching have been implemented in order to address the ever increasing number of triples in Virtuoso and improve the performance of the YDS Open Data Repository and the YDS platform.

The first layer is within the mu-cl-resources microservice itself, designed to serve future requests faster by creating a cache at the level of objects and their properties. Additionally, a separate caching microservice, named *mu-cache*, has been developed as a reusable component in the mu-semte-ch architecture.

3.2.2.1 Mu-cache

Detecting when to clear the cache on a URL-basis is near-impossible, given that the set of possible calls in a JSONAPI endpoint is non-exhaustive. In some configurations, the set of URLs which yield a particular resource may be infinite. Primitives for managing such a cache are needed.

Managing the cache within a specific microservice is probably not wanted as the microservice would need to communicate with its peers to indicate which items are in the cache. For instance, if we have two mu-cl-resources containers using the same image, we could load-balance between these instances. When one of these detects an update to the model, the other would need to clear its cache also.

The mu-cache provides a distributed caching proxy for JSON API like resources. The service can be placed in front of any microservice which understand the cache's primitives. Microservices can update the mu-cache within their standard request-response cycles or by making REST calls on the mu-cache (REST calls are still under development).

3.2.3 Implementation of the model and extensions

The most generic JSON responses are offered by mu-cl-resources. This component offers a JSON API compatible API for the content in the triplestore. It also allows for the updating of the contents in the triplestore if the SPARQL endpoint allows for write operations.

The mu-cl-resources microservice needs to be configured so that it knows how to map the contents of the triplestore to a JSON API compatible response. Configuration occurs through a paren-based syntax. In order to support a catalog resource of the DCAT standard, one would use the following snippet:

```
(define-resource catalog ()
  :class (s-prefix "dcat:Catalog")
  :properties `((:title :string ,(s-prefix "dct:title"))
              (:description :string ,(s-prefix "dct:description"))
              (:issued :string ,(s-prefix "dct:issued"))
              (:modified :string ,(s-prefix "dct:modified"))
              (:language :string ,(s-prefix "dct:language"))
              (:license :string ,(s-prefix "dct:license"))
```

```
               (:rights :string ,(s-prefix "dct:rights"))
               (:spatial :string ,(s-prefix "dct:spatial"))
               (:homepage :string ,(s-prefix "foaf:homepage")))
  :has-one `((publisher :via ,(s-prefix "dct:publisher")
                   :as "publisher")
             (concept-scheme :via ,(s-prefix "dcat:themeTaxonomy")
                   :as "theme-taxonomy")
             (catalog-record :via ,(s-prefix "dcat:record")
                   :as "record"))
  :has-many `((dataset :via ,(s-prefix "dcat:dataset")
                   :as "datasets"))
  :resource-base (s-url "http://your-data-stories.eu/catalogs/")
  :on-path "catalogs")
```

This ensures there is a catalog resource hosted on "/catalogs" which contains entries of type dcat:Catalog. The entries have 9 different properties, and 4 relationships. If the YDS data model changes, we would alter this snippet and the system would consume and produce a different model. This approach allows us to be flexible with regards to the model.

The mu-cl-resources microservice offers support for declaratively specifying resources, and it offers a full JSON API compliant API for the specified resources.

3.3 Extensions

The first version of the Open Data Repository provided a SPARQL endpoint as well as a JSON API endpoint, to support Linked Data experts and developers without Semantic Web expertise alike. The assumption was that Linked data aficionados as well as people that expect full control over the returned data will likely prefer the SPARQL endpoint, whereas future driven web developers will likely use JSON API as their tool of choice. However, while SPARQL provided power and flexibility, it lacked the simplicity and ease of use of a RESTful API. Hence, we decided to bridge the gap and introduce a third channel for accessing the YDS Open Data Repository, the JSON-LD API.

3.3.1 JSON-LD API

JSON-LD (JavaScript Object Notation for Linked Data) is designed around the concept of a "context" to deliver a valid JSON representation of an RDF model. Responses are accompanied by the mappings necessary to expand properties to IRIs which makes it possible to retrieve even more information, in a true Linked Data spirit. The API overcomes a number of shortcomings of the JSON API, which was never intended for the Web of Data, while still providing a Web developer friendly way to access the YDS repository.

3.3.2 Apache Solr

As previously noted, even though OpenLink Virtuoso is one of the most powerful triple stores on the market today, it is not an omnipotent storage solution. One of most common weaknesses of RDF databases is natural language search, which is reflected in both poor performance and flexibility for non-experts.

To provide advanced full-text search capabilities, such as phrases, wildcards, joins, and grouping, across any data type, all natural language fields are indexed in Apache Lucene, the underlying engine in Apache's standalone enterprise search platform, Solr. Solr is scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, and centralized configuration. Moreover, it provides a REST-like API, and is used to power the YDS platform's search and navigation features being developed in WP4.

4 Available APIs

All content can be retrieved through the SPARQL endpoint, which can return query responses in several different formats. The latest version of the data model has been defined in D3.4 and the SPARQL query language is fully described in SPARQL 1.1 [1]. We don't go in more depth on the use of SPARQL in this deliverable as it is mostly suited for Linked Data experts who are already familiar with the technology.

4.1 JSON API

The JSON API compliant specification supports all resources of the current data model. It also supports pagination and filtering. We supply some examples so an initial impression can be formed on the responses, but urge the interested reader to read up on [2].

All resources are hosted top-level. The related requests for a particular request are supplied in the top-level links set of the response. Each resource has a self, which can be used to retrieve that specific resource. If we retrieve a listing of catalogs, the following request/response could occur:

```
REQUEST /catalogs

RESPONSE
{
 "data": [
  {
   "attributes": {
    "title": "YDS example catalog",
    "description": "A catalog which is built as an example for D3.9, D3.10 and D3.11"
   },
   "id": "4646989B-D91A-4685-AA2F-03EBEC06CE8F",
   "type": "catalogs",
   "relationships": {
    "datasets": {
     "links": {
      "self": "/catalogs/4646989B-D91A-4685-AA2F-03EBEC06CE8F/links/datasets",
      "related": "/catalogs/4646989B-D91A-4685-AA2F-03EBEC06CE8F/datasets"
     }
    },
    "publisher": {
     "links": {
      "self": "/catalogs/4646989B-D91A-4685-AA2F-03EBEC06CE8F/links/publisher",
      "related": "/catalogs/4646989B-D91A-4685-AA2F-03EBEC06CE8F/publisher"
     }
    },
    "theme-taxonomy": {
     "links": {
      "self": "/catalogs/4646989B-D91A-4685-AA2F-03EBEC06CE8F/links/theme-taxonomy",
            "related": "/catalogs/4646989B-D91A-4685-AA2F-03EBEC06CE8F/theme-taxonomy"
          }
        },
        "record": {
          "links": {
            "self": "/catalogs/4646989B-D91A-4685-AA2F-03EBEC06CE8F/links/record",
            "related": "/catalogs/4646989B-D91A-4685-AA2F-03EBEC06CE8F/record"
          }
        }
      }
    }
  ],
  "links": {
    "last": "/catalogs&page[number]=2",
    "first": "/catalogs",
    "next": "/catalogs&page[number]=1"
  }
}
```

Not all attributes were specified in the database. The attributes which are specified and described will be returned. If a links object is specified at the top-level, it contains references to related links. This could be for pagination, or for finding a (set of) related resource(s).

Search/filtering of responses is supported too, but is still subject to change. How filtering evolves, will depend on the uses, as they show up in the developed YDS components, as well as the evolution of the JSON API specification. JSON API recommends filtering in an open-ended way. We provide a few examples in this follow up to make our choices simpler to digest.

The simplest form of filtering is to filter on an attribute. We could search for all catalogs with YDS in the title by querying the path /catalogs?filter[title]=YDS. Filter paths can be used too. Searching for all catalogs which contain a dataset which contains the word *"links"* in the description could be done by issuing /catalogs/filter[datasets][description]=links. You can supply more than one filter in a request, all conditions must hold in that case. Filtering can also occur on the identifier of a linked resource. In order to find all datasets which contain both the theme with identifier *"ABC"* as well as the theme with identifier *"DEF"*, one could search for /datasets?filter[themes]=ABC&filter[themes]=DEF.

Sorting response data via a *sort* query parameter is also supported. For example, we could sort all aid activities by the project title, by querying the path /aid-activities?sort=title. Multiple sort fields can also be used, by merely separating them with a comma, e.g. /aid-activities?sort=title,project-id.

Every endpoint returns the top level data, with links to related resources, but selected links can be expanded/included in the response by using the *include* parameter. For instance, including a benefactor in a request for aid activities would correspond to /aid-activities?include=benefactor. In order to request resources related to other resources, a dot-separated path for each relationship name can be specified as follows: /aid-activities?include=benefactor.country.

A user can also request that an endpoint return only specific fields in the response on a per-type basis by including a fields[TYPE] parameter. The value of the fields parameter must be a comma-separated list that refers to the name(s) of the fields to be returned. Take, for example, the extended path shown earlier: /aid-activities?include=benefactor&fields[aid-activities]=title,description.

The content which is offered through the JSON API is easy to discover once a start point has been found. An element of future work would be to auto-document this specification. This would yield a large document which could be used for discovering what is available in the repository. Reading the declarative specification used by mu-cl-resources will help until then.

- Mu-semte-ch improvements
    - Language tags
    - Inclusion of URIs

4.2 JSON-LD API

The JSON-LD API supports all resources of the current data model (described through the JSON-LD[6] specification). It also supports the user interface components implemented in WP4, the replication of all resources into Solr, advanced search capabilities (through Solr), spatial search, and supports the workbench, also implemented in WP4. The available endpoints can be classified into the following categories:

- Returning resources in JSON-LD. This category includes the following endpoints:
    - model/describe.tcl
    - model/class.tcl
    - model/classes.tcl
    - model/dataset.tcl
- Supporting WP4 visualisation components (charts)
    - component/barchart.tcl
    - component/bubblechart.tcl
    - component/grid.tcl
    - component/heatmap.tcl
    - component/linechart.tcl
    - component/map.tcl
    - component/piechart.tcl
    - component/scatterchart.tcl
    - component/treemap.tcl
- Supporting search
    - component/search.tcl
    - component/searchadvanced.tcl
    - component/spatialsearch.tcl
    - component/suggest.tcl
    - component/suggestfield.tcl

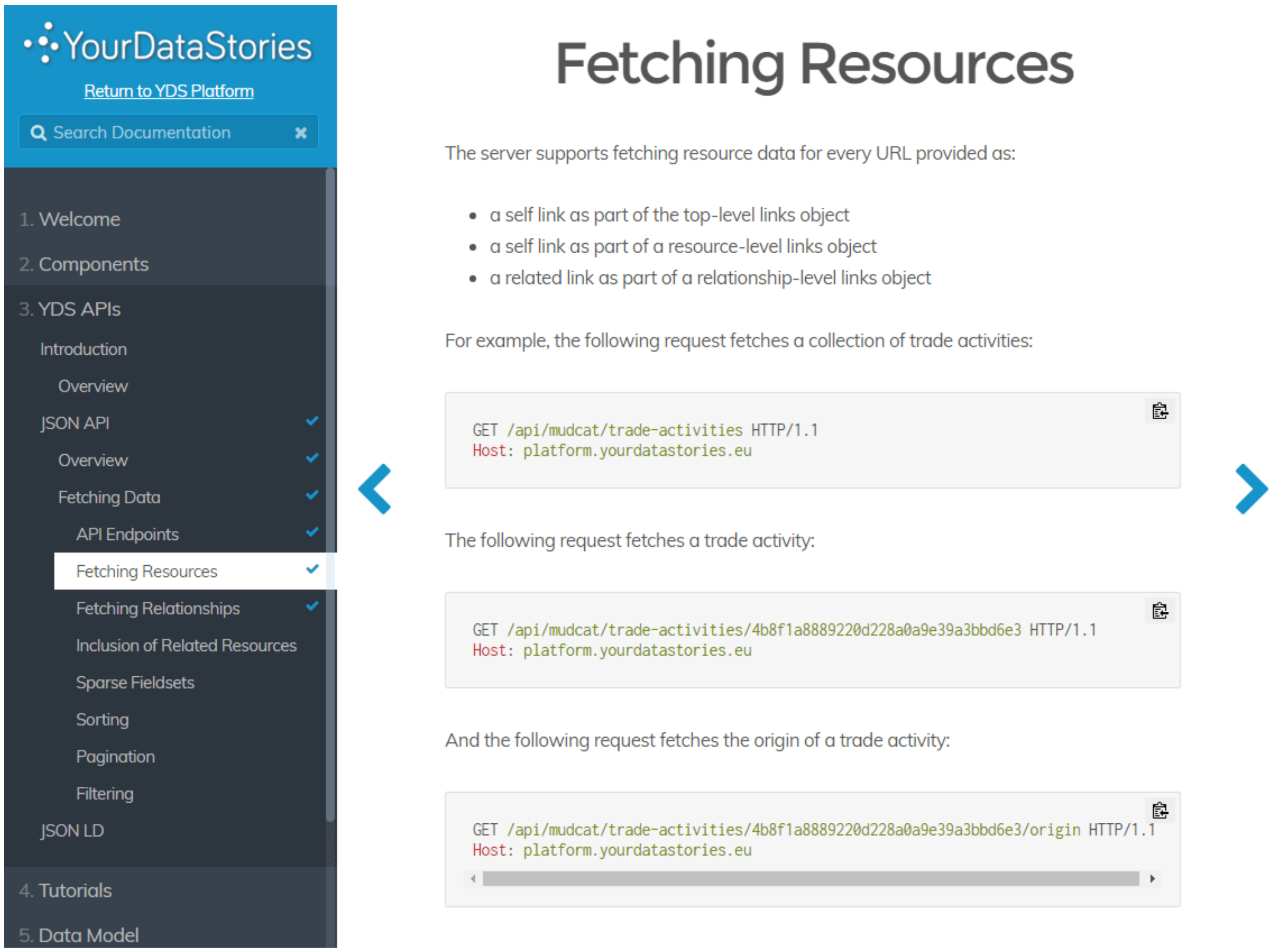
[6] JSON-LD – JSON for Linked Data: http://json-ld.org/.

    - component/type2advancedquery.tcl
    - component/type2solrquery.tcl
- Supporting workbench and other visualisation components
    - component/aggregate.tcl
    - component/info.tcl
    - component/statistics.tcl
    - component/filter.tcl
    - component/filters.tcl
    - component/plotinfo.tcl

For the sake of brevity in this deliverable, the complete documentation of all endpoints is provided as an Open API compliant specification, as described in Section 1.

5 Developer documentation

The complete, up-to-date developer documentation for both APIs is provided online. The JSON API documentation is, for the time being, available on the YDS servers, whereas the Open API compliant JSON-LD API specification is provided on SwaggerHub, in hope of increasing its visibility.



Fetching Resources

The server supports fetching resource data for every URL provided as:

- a self link as part of the top-level links object
- a self link as part of a resource-level links object
- a related link as part of a relationship-level links object

For example, the following request fetches a collection of trade activities:

```
GET /api/mudcat/trade-activities HTTP/1.1
Host: platform.yourdatastories.eu
```

The following request fetches a trade activity:

```
GET /api/mudcat/trade-activities/4b8f1a8889220d228a0a9e39a3bbd6e3 HTTP/1.1
Host: platform.yourdatastories.eu
```

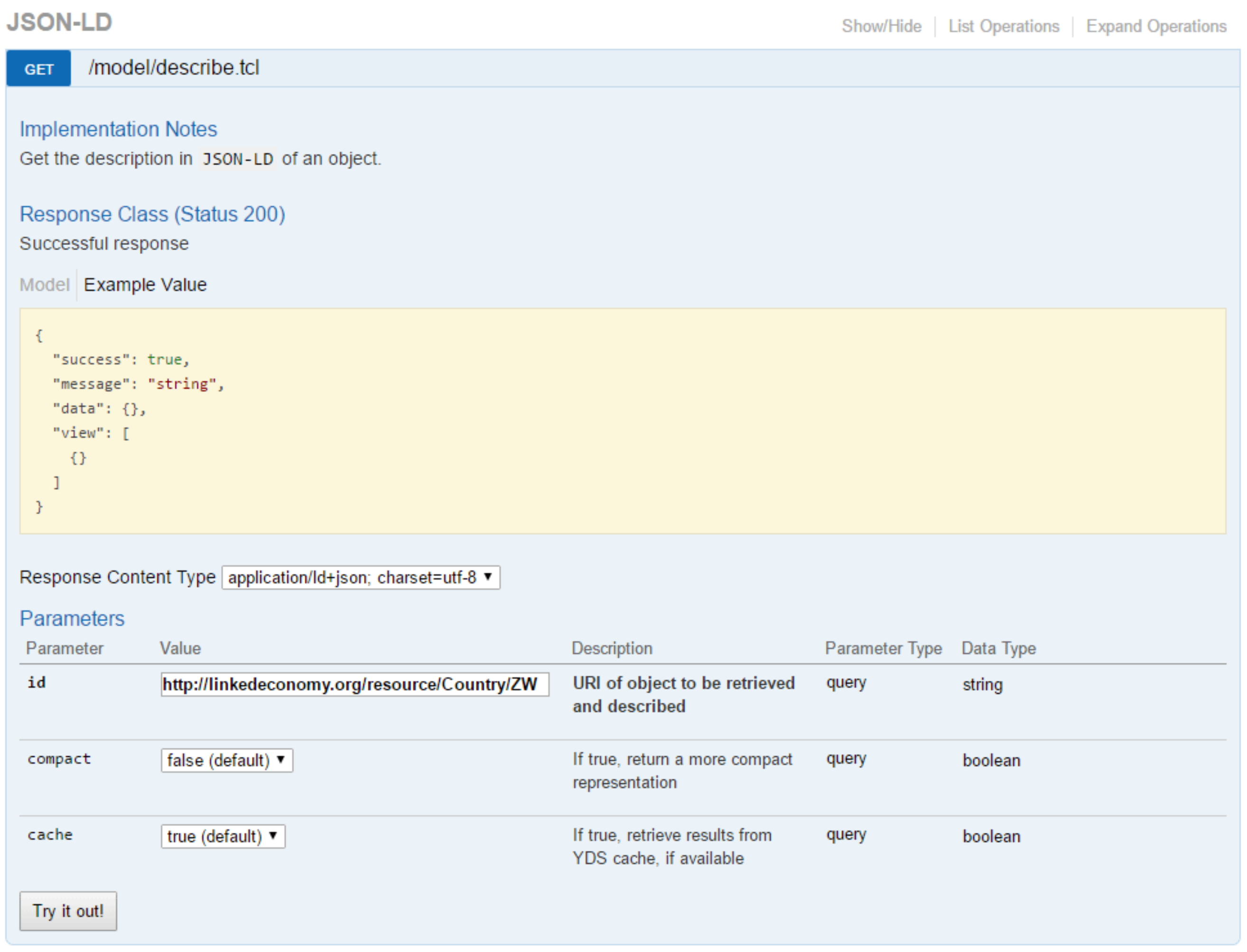
And the following request fetches the origin of a trade activity:

```
GET /api/mudcat/trade-activities/4b8f1a8889220d228a0a9e39a3bbd6e3/origin HTTP/1.1
Host: platform.yourdatastories.eu
```

Figure 3: JSON API Documentation

As the mu-cl-resources configuration is rather complex, and changes as often as the model itself, maintaining an Open API specification manually would be expensive, which is why we intend to automate the generation of the JSON API documentation in the future. In the meantime, the developer's manual can be accessed via the following URL:

http://ydsdev.iit.demokritos.gr/YDS-docs/API/JSON-API/Fetching-Data



JSON-LD

/model/describe.tcl

Implementation Notes

Get the description in `JSON-LD` of an object.

Response Class (Status 200)

Successful response

Example Value

```
{
  "success": true,
  "message": "string",
  "data": {},
  "view": [
    {}
  ]
}
```

Response Content Type: application/ld+json; charset=utf-8

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	http://linkedeconomy.org/resource/Country/ZW	URI of object to be retrieved and described	query	string
compact	false (default)	If true, return a more compact representation	query	boolean
cache	true (default)	If true, retrieve results from YDS cache, if available	query	boolean

Figure 4: JSON-LD API interactive documentation on SwaggerHub

All JSON-LD API endpoints are documented and made publicly available on SwaggerHub, an all-in-one collaborative API documentation, validation, and testing platform:

https://app.swaggerhub.com/api/yourdatastories.ncsrd/your-data_stories_repository_api_json_ld_and_components/2.0.1

It is also worth noting that the full, up-to-date data model documentation is also provided on GitHub[7] and further described in D3.4.

[7] https://github.com/YourDataStories/ontology

6 Future Vision

This is the second release of the Open Data Repository. The final release will focus further on performance and extending the capabilities of the provided APIs.

Among the most important improvements we foresee in the last year of the project is the support for user authorization. This will allow finer-grained access control, as well as prevent potential abuse of the API and the server resources.

Second, we intend to further extend the mu-semte-ch architecture and implement even more features, such as the support for querying data ranges (e.g. amounts or dates). As such extensions might fall out of the JSON API spec, these additions might come in the form of new microservices.

Moreover, as mentioned earlier, we will consider the possibility of automatic Open API specification compliant documentation generation for mu-cl-resources. This would ensure easier documentation maintenance, a standardized documentation format, and, ultimately, providing up to date documentation to end users. The format will also be aligned across all YDS Open Data Repository APIs, and placing all documentation on SwaggerHub will increase the API and overall project visibility.

Additionally, even though the YDS platform currently serves as the most powerful visual repository browser, we will investigate the possibility of providing web developers with a lightweight visual interface for mu-cl-resources.

Finally, we will focus on fine-tuning, clean-up and error-fixing to ensure the final version of the YourDataStories Open Data Repository is delivered stable and bug-free.

7 Conclusion

The first version of the Open Data Repository was based on user-facing microservices which store their data in a shared triplestore. It provided support for retrieving the contents either through JSON API, or through SPARQL. This was further extended through the addition of new microservices, but also an enterprise search engine and a JSON-LD specification compliant API. Existing microservices have also been improved via new features. All API channels have been documented and made available online to future developers.

8 References

[1] SPARQL 1.1 Overview <http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/>
[2] JSON API <http://jsonapi.org>
[3] Mu-semtech <http://github.com/mu-semtech>
[4] A. Versteden, E. Pauwels, and A. Papantoniou. *An Ecosystem of User-facing Microservices supported by Semantic Models*. The 5th International USEWOD Workshop, Portoroz, Slovenia. 2015. <http://usewod.org/files/workshops/2015/papers/USEWOD15_versteden_pauwels_papantaniou.pdf>
[5]Mu-cl-resources <https://github.com/mu-semtech/mu-cl-resources>
[6] Open Link Virtuoso <http://virtuoso.openlinksw.com/>
[7] JSON <http://json.org>
[8] Docker <https://www.docker.com/>
[9] Docker Compose <https://docs.docker.com/compose/>
[10] Mu-dispatcher <https://github.com/mu-semtech/mu-dispatcher>
[11] Mu-identifier <https://github.com/mu-semtech/mu-identifier>
[12] Mu-cache <https://github.com/mu-semtech/mu-cache>
[13] JSON-LD 1.0 <https://www.w3.org/TR/json-ld/>
[14] Apache Solr <http://lucene.apache.org/solr/>