

Social Sciences & Humanities Open Cloud

Project Number: 823782

Start Date of Project: 01/01/2019

Duration: 40 months

Deliverable 7.2 Marketplace – Implementation

Dissemination Level	PU
Due Date of Deliverable	31/12/2021 (M36)
Actual Submission Date	17/11/2021
Work Package	WP 7 - Creating the SSH Open Marketplace
Task	Task 7.2 Development of the Marketplace Application
Type	Report
Approval Status	Waiting EC approval
Version	V1.0
Number of Pages	p.1 – p.50

Abstract:

Report by Task 7.2 team accompanying the implementation and final public release version of the SSH Open Marketplace application on the background of the system specification published as *D7.1 System Specification - SSH Open Marketplace*, in September 2019.

The information in this document reflects only the author’s views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided “as is” without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/ her sole risk and liability. This deliverable is licensed under a Creative Commons Attribution 4.0 International License.



History

Version	Date	Reason	Revised by
0.0	06/2021	Skeleton	Frank Fischer, Laure Barbot
0.1	09/2021	First draft	Matej Ďurčo, Laure Barbot with contributions from Klaus Illmayer, Seung-Bin Yim, Sotiris Karampatakis, Frank Fischer, Yoann Moranville, Joshua Tetteh Ocansey, Stefan Probst, Michał Kozak and Stefan Buddenbohm.
0.2	10/2021	Review	Stefania Martziou, Antonis Lempesis (OpenAire) & John Shepherdson (CESSDA)
1.0	15/11/2021	Addressing comments	Matej Ďurčo, Laure Barbot

Author List

Organisation	Name	Contact Information
DARIAH/OEAW	Matej Ďurčo	matej.durco@oeaw.ac.at
DARIAH	Laure Barbot	laure.barbot@dariah.eu
DARIAH/OEAW	Klaus Illmayer	klaus.illmayer@oeaw.ac.at
SWC	Sotiris Karampatakis	sotiris.karampatakis@semantic-web.com
DARIAH	Frank Fischer	frank.fischer@dariah.eu
DARIAH	Yoann Moranville	yoann.moranville@dariah.eu
CESSDA	Joshua Tetteh Ocansey	joshua.ocansey@cessda.eu
DARIAH/OEAW	Stefan Probst	Stefan.Probst@oeaw.ac.at
DARIAH/PSNC	Michał Kozak	mkozak@man.poznan.pl
DARIAH/UGOE	Stefan Buddenbohm	buddenbohm@sub.uni-goettingen.de
DARIAH/OEAW	Seung-Bin Yim	Seung-Bin.Yim@oeaw.ac.at

Executive Summary

This document delivers the results of Task 7.2 “Development of the Marketplace application” of the Social Sciences and Humanities Open Cloud (SSHOC) project funded by the European Commission under Grant Agreement #823782. Its main purpose is to describe the actual implementation of the SSH Open Marketplace application on the background of the system specification (delivered in 2019 as *D7.1 System Specification - SSH Open Marketplace*).

The SSH Open Marketplace is a discovery portal which pools and contextualises resources for Social Sciences and Humanities research communities: tools, services, training materials, datasets and workflows. The Marketplace highlights and showcases solutions and research practices for every step of the SSH research data life cycle.

Based on an agile methodology and an iterative process of releases, the implementation of the SSH Open Marketplace is carefully described here, and this deliverable represents a companion to the final release of the application (in December 2021). Three other complementary WP7 deliverables published between September and December 2021 are also documenting and supporting the final release: *D7.3 Marketplace Interoperability*; *D7.4 Marketplace Data population & curation* and *D7.5 Marketplace Governance*.

Following the requirements engineering process and the first iterations of the data model and system architecture presented in the System Specification, this implementation report describes the methodology used to ensure the quality of the SSH Open Marketplace software, and presents the latest iteration of the data model and system architecture supported by the rationales for the implementation choices. Key components and features, as well as other aspects of the system like its configuration, deployment or its API are described giving a technical insight into the creation of this discovery portal.

Embedded in the European Open Science Cloud (EOSC) ecosystem, the SSH Open Marketplace has also been developed to contribute to the resources discoverability layer of the EOSC and to ensure, thanks to its user-friendly interface, an entry door to the EOSC for social scientists and humanists.

Abbreviations and Acronyms

ACDH-CH	Austrian Centre for Digital Humanities and Cultural Heritage
API	Application Programming Interface
CESSDA	Consortium of European Social Science Data Archives
CLARIN	Common Language Resources and Technology Infrastructure
DARIAH	Digital Research Infrastructure for the Arts and Humanities
DPU	Data Processing Units
EOSC	European Open Science Cloud
ETL	Extract, Transform, Load
EURISE	European Research Infrastructure Software Engineers
GWDG	Society for Scientific Data Processing
IdPs	identity providers
JSON-LD	JavaScript Object Notation for Linked Data
JWT	JSON Web Token
LOD	Linked Open Data
M	month
MP	(SSH Open) Marketplace
MS	milestone
NLP	Natural Language Processing
OEAW	Austrian Academy of Sciences
ORCID	Open Researcher and Contributor ID
PSNC	Poznan Supercomputing and Networking Center
RDF	Resource Description Framework
REST API	representational state transfer Application Programming Interface
RML	RDF Mapping Language
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
SSH	Social Sciences and Humanities
SSHOC	Social Sciences and Humanities Open Cloud

SWC	Semantic Web Company
T	task
UV	UnifiedViews
UX	User eXperience
WP	work package

Table of Contents

Introduction	7
Methodology	8
Development guidelines	8
Communication	8
GitLab for code maintenance & issue tracking	9
Data Model	11
Main features of the data model	12
SSH Open Marketplace data model in context	16
Implementation	18
From specification to implementation	18
Backend component	23
Frontend component	24
Ingestion	25
Curation Components	32
Extraction module	38
Deployment and configuration of the system	40
Application Programming Interface	42
EOSC Integration	42
EOSC AAI	43
Onboarding SSH Open Marketplace into the EOSC portal	44
Research community portals in the EOSC	45
Conclusion	46
References	47
Annex 1 - EURISE Software Quality Checklist applied to the SSH Open Marketplace	49

1. Introduction

The SSH Open Marketplace is a discovery portal revolving around five general types of content: tools & services, training materials, workflows, datasets and publications. Thanks to a contextualisation layer (relations between items and research workflows exemplifying the use of some items), the SSH Open Marketplace helps to increase the discoverability and findability of useful resources to support the uptake of digital methods in Social Sciences and Humanities.

This report describes the actual implementation of the SSH Open Marketplace application on the background of the system specification published in September 2019¹. It summarises the development method and work carried out to achieve the three releases of the application over the course of the SSHOC project: the alpha release in June 2020², the beta release in December 2020³ and the final release planned for December 2021. As the present report is submitted a few months before the final release, Milestone 44 report that will be published at the beginning of 2022 might include some final implementation remarks, but at the time of writing, most of the remaining issues are already identified, prioritised and assigned to ensure successful completion of the final iteration of the application.

Though the actual technical implementation work was within task (T) 7.2, given that all tasks of the work package were geared towards one common main outcome, there was a strong cohesion and intensive exchange between the four tasks of the work package⁴, basically forming one team with smaller groups being dynamically formed to attend to individual issues at hand.

After a first chapter presenting the underlying methodology used to develop the SSH Open Marketplace, the data model and the system architecture of the application are described in the light of the changes made since the specification phase. Decisions taken during the implementation phase by the task 7.2 members, informed by input from the work of tasks 7.3 and 7.4, are described and motivated. Finally, a chapter about the relation of SSH Open Marketplace to European Open Science Cloud (EOSC) is added in order to highlight the specificities of the landscape in which the SSH Open Marketplace is developed.

¹ Laure Barbot, Yoan Moranville, Frank Fischer, Clara Petitfils, Matej Ďurčo, Klaus Illmayer, Tomasz Parkoła, Philipp Wieder, & Sotiris Karampatakis. (2019). SSHOC D7.1 System Specification - SSH Open Marketplace (1.0). Zenodo. <https://doi.org/10.5281/zenodo.3547649> [21.09.2021]

² Laure Barbot, Yoann Moranville, Stefan Buddenbohm, Klaus Illmayer, & Matej Ďurčo. (2020). MS42 Marketplace – alpha release (v1.0). Zenodo. <https://doi.org/10.5281/zenodo.4585700> [22.09.2021]

³ Laure Barbot, Frank Fischer, Klaus Illmayer, Matej Ďurčo, Alexander König, Dieter Van Uytvanck, & Nicolas Larrousse. (2020). MS.43 -Marketplace - beta release (1.0). Zenodo. <https://doi.org/10.5281/zenodo.4785194> [22.09.2021]

⁴ Task 7.1 User requirements, Conceptual Model and System Architecture of the SSH Open Marketplace; task 7.2 Development of the Marketplace Application; task 7.3 Marketplace Interoperability; task 7.4 Governance: Population, Curation & Sustainability of the SSH Open Marketplace.

2. Methodology

2.1. Development guidelines

As described in *D7.1 System Specification*, the SSH Open Marketplace (MP) has been developed following several key principles of the agile software development methodology. Requirements and implementation evolved during the three years' development period, based on a User Centered Design approach, supported by SSHOC WP2 and WP6 dissemination and event teams and driven by UX designer from DARIAH/PSNC, that allowed T7.2 developers led by DARIAH/OEAW⁵ to refine the key functionalities of the application. An overview of the different methods used to ground development work on SSH researchers and future users needs can be found in MS43 report⁶.

Furthermore, within the SSHOC project some recommendations and guidelines for software development⁷ have been shared by the coordinator CESSDA in order to ensure the quality and the coherence of the different outputs of the project. The EURISE Technical Reference⁸, jointly developed by CESSDA, CLARIN & DARIAH, and their guidelines have been a good starting point for T7.2 activities. The EURISE software quality checklist has been added in Annex 1 and summarises the steps taken to ensure quality of the SSH Open Marketplace as a software.

The developed code is released under APACHE-2.0 open source license⁹.

2.2. Communication

In order to ensure appropriate communication between the T7.2 team members, with the rest of the project and with the outside world, several communication channels have been used:

- Regular (bi-weekly or monthly) tele-meetings for all WP7 members, and for T7.2 members on demand either focusing on design and frontend implementation or gathering all developers involved in the task.

⁵ The institute from OEAW involved in the project and leading T7.2 is the Austrian Centre for Digital Humanities and Cultural Heritage (ACDH-CH).

⁶ A combination of live events, consultation platform and guerrilla interviews was set up to ensure that the initial user requirements and specifications remained pertinent and coherent over the course of their implementation.

⁷ See the internal documentation:

<https://sites.google.com/cessda.eu/sshocwiki/collaboration/software-development> [13.10.2021]

⁸ See EURISE Technical Reference: <https://technical-reference.readthedocs.io> [22.09.2021]

⁹ More information about this license can be found here:

<https://choosealicense.com/licenses/apache-2.0/> [13.10.2021]

- A dedicated Basecamp project for SSHOC WP7 has been used to follow high-level tasks derived from the Description of Work.
- Multiple Gitlab repositories, for maintaining the code and more fine-grained issues related to the implementation of individual components of the overall system, have been set up (see next section).
- A dedicated folder for WP7 within the Shared Drive procured by the project coordinator for the whole project has been used for collaborative authoring of project documents and sharing of (non-implementation) files.
- A dedicated Slack channel within the DARIAH namespace `dariah.slack.com` #marketplace has been opened for quick ad hoc communications of the development team.

In specific cases - such as the data model update described in the next chapter - specific communication workflows involving all parties have been agreed, documented and maintained through dedicated “tracking changes” or “umbrella” issues¹⁰ in Gitlab.

2.3. GitLab for code maintenance & issue tracking

To maintain code and other technical resources as well as the implementation-related issues, a dedicated group¹¹ within the gitlab instance provided by DARIAH/UGOE¹² has been used, divided into multiple projects corresponding to the component boundaries of the system:

- <https://gitlab.gwdg.de/sshoc/sshoc-marketplace>
General project for the overall division of work, especially holding the set of requirements distilled in the initial phase of the project expressed as individual issues.
- <https://gitlab.gwdg.de/sshoc/sshoc-marketplace-frontend/>
code and issues related to the frontend component
- <https://gitlab.gwdg.de/sshoc/sshoc-marketplace-backend>
code and issues related to the backend component
- <https://gitlab.gwdg.de/sshoc/vocabularies>
project with repository to keep versions of vocabularies used in the Marketplace serialized in SKOS and accompanied by vocabulary-related issues
- <https://gitlab.gwdg.de/sshoc/data-ingestion>
scripts and auxiliary files pertaining to the ingestion procedures: RML¹³ mappings, Unified Views pipelines and other configuration files

¹⁰ See for example the “Tracking changes to dynamic properties” issue used to inform tasks 2, 3 and 4 of the latest changes in dynamic properties implementation: <https://gitlab.gwdg.de/sshoc/sshoc-marketplace/-/issues/66> [13.10.2021]

¹¹ SSHOC group in GWDG Gitlab instance: <https://gitlab.gwdg.de/sshoc> [22.09.2021]

¹² More precisely, the GWDG.

¹³ RML is the RDF Mapping Language while RDF stands for Resource Description Framework.

- <https://gitlab.gwdg.de/sshoc/curation>
scripts and auxiliary data for the ongoing curation of the Marketplace

The development team made extensive use of various features provided by gitlab allowing to structure the work and foster the team communication in accordance with the principles of agile software development. A set of custom tags or labels have been introduced to categorize the issues according to their status (ToDo, InDev, InReview, etc.) and the area they pertain to (data-model, content, etc.). Milestones¹⁴ were used to group issues that needed to be resolved to achieve a certain state, examples being: “Minimal viable product”, “Alpha release”, “Curation module implemented”, “Final release”. Kanban boards also helped the team to keep track of the progress of issues.

¹⁴ See Milestones board: <https://gitlab.gwdg.de/groups/sshoc/-/milestones> [13.10.2021]

3. Data Model

The SSH Open Marketplace Data Model was first described in the *D7.1 System Specification* document¹⁵. Since the publication of D7.1 in M9, the data model has undergone a series of iterations, guided by the practical needs of the implementation and the actual data from external sources the team was dealing with. The most recent version of the data model, 1.5, is depicted in the diagram below figure 1.

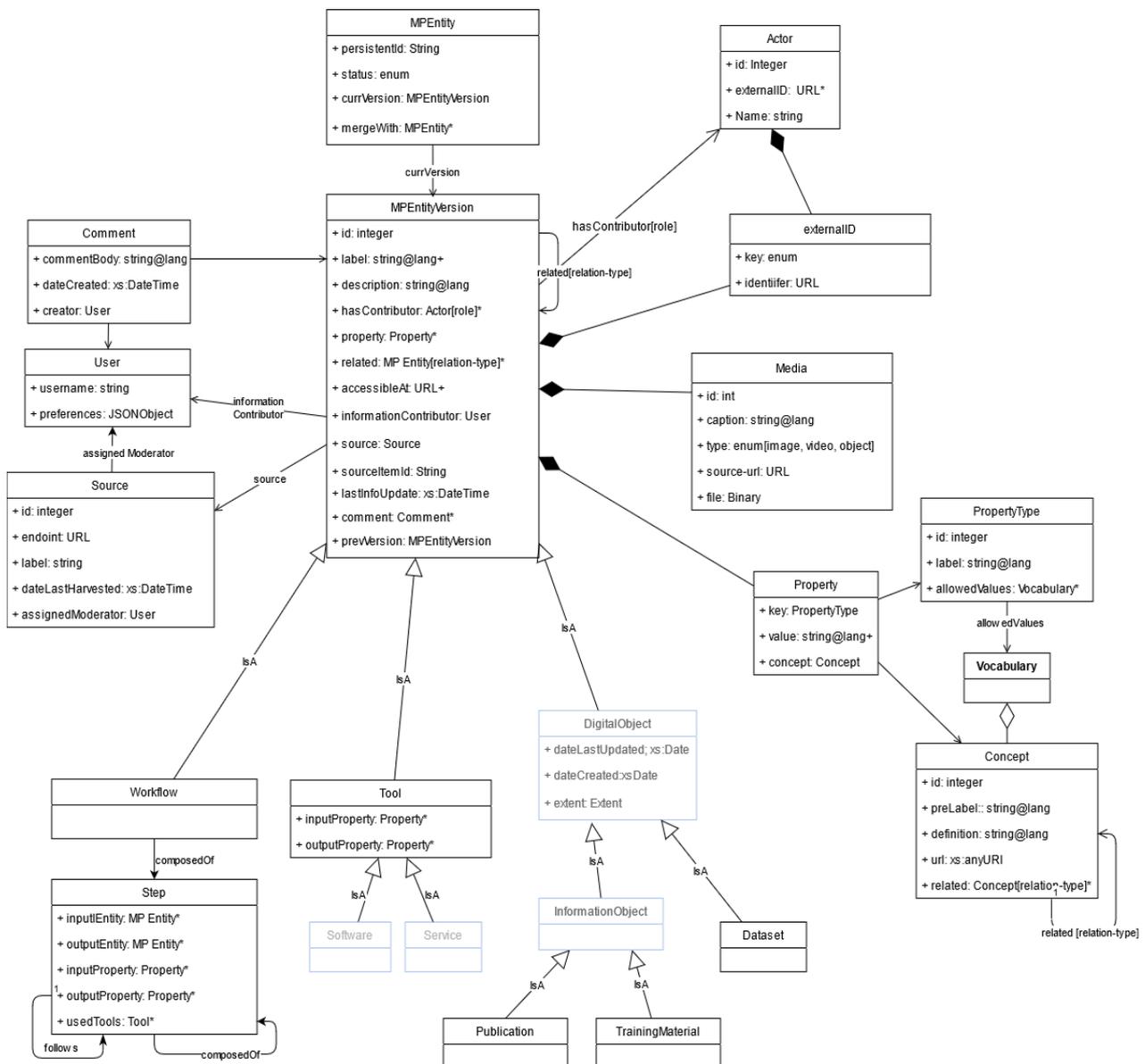


Fig. 1 - Data Model v 1.5

¹⁵ SSHOC D7.1 System Specification - SSH Open Marketplace, p. 24 and following.

3.1. Main features of the data model

In the following paragraphs the main features of the data model are described (features introduced since version 1.0 proposed in D7.1 are marked with *).

MPEntity and main categories

The main class of the data model is *MPEntity*. All content items of the SSH Open Marketplace are instances of *MPEntity*, though indirectly by means of one of the subclasses. In contrast to the original class hierarchy based on conceptual considerations, the final system implements a simplified set of main classes as dictated by the practical understanding and needs of the users. These are: *Tools&Services*, *Publication*, *Dataset*, *TrainingMaterial*, and *Workflow*, omitting originally introduced intermediary classes *DigitalObject*, and *InformationObject*, as well as convoluting *Software* and *Service* to one class, *Tool*¹⁶. These main classes are also reflected as dedicated endpoints in the systems API, and thus considered very stable structures, where a major refactoring of the application would be required, if these were to change.

The *MPEntity* class also implements most of the generic aspects of the data model: versioning, dynamic properties, relations, media, external identifiers, commenting, provenance.

Flexible classification/typing

The above argument also implies that the data model should not hardwire a rich class hierarchy. Therefore, most information about the described resource is captured already in the top class *MPEntity* and the majority of classification/typing is expected to be covered by appropriate Properties with corresponding Vocabularies. As an example, the various types of training materials, as expressed in the user requirements, are modelled as a property of an *MPEntity* instance, rather than being hardwired as classes in the data model. This is because a) there is no global consensus on the categorisation, b) a training material could belong to multiple categories, c) from the system point of view there is no principle difference between the different types, and it is more efficient to handle them all uniformly as *MPEntities*.

Generic model

Given the heterogeneous dataspace that the service covers, and the broad and underspecified range of information to be captured about the entities represented in the MP, a generic data model that can be refined through configuration during runtime has been chosen. The main mechanisms are: every *MPEntity* can have a set of *Properties*, i.e. key-value pairs, where allowed keys and allowed values for individual keys are specified in the configuration of the system, not in the data model itself. Equally, there is a generic relation between *MPEntities*, that can be typed as needed.

¹⁶ In this case, there is a slight inconsistency between the internal class system and the exposed API, where this entity type is referred to as *tools-services*.

Versioning*

To support the envisaged curation workflow and to ensure complete traceability, a comprehensive versioning system has been put in place that retains a complete record of all changes to the data, including the author of the change. This allows the curator to see, compare and approve/dismiss changes proposed by editors, or introduced by automatic updates. To this end, a new generic class *MPEntityVersion* has been introduced next to the main class *MPEntity*, with multiple versions being attached (1:N relation) to one *MPEntity*, and at most one version being the current approved version. The most complex part of the versioning system is tracking the changes to relations, because these pertain not just to the edited item, but also the related one. An intricate mechanism has been put in place, where also the version of the related items is bumped if there is a change to the relation.

The moderators mode of the frontend shows a version history, allowing the Moderators to review previous versions and even to revert changes.

Status*

The implementation of versioning prompted a need for an elaborate status handling of individual versions. The following statuses have been introduced:

Statuses for items:

- DRAFT - instead of immediately publishing an item on the MP, users can create a draft version of an item. A draft item is accessible only by the user who creates it and cannot be shared with other users. The user can add further information and publish the item when it is ready, thus setting the new state to "SUGGESTED" (if the user is a Contributor) or "APPROVED" (if the user is a Moderator/Administrator).
- INGESTED - status for items coming from an ingestion pipeline. This allows Moderators/Administrators to differentiate them from items that are "SUGGESTED" by Contributors. These ingested items are passed on to the Moderators/Administrators for approval/rejection.
- SUGGESTED - status of items created or updated by Contributors. These items are submitted to Moderators/Administrators for approval/rejection.
- APPROVED - items with the status "INGESTED" or "SUGGESTED" get the status "APPROVED" when they have been reviewed and accepted by a Moderator/Administrator.
- DISAPPROVED - items with the status "INGESTED" or "SUGGESTED" get the status "DISAPPROVED" if they have been rejected by a Moderator/Administrator.
- DEPRECATED - items that are deleted by Administrators. They are not visible to the public in the frontend anymore. Deleted items are still available for authorized accounts - e. g. an ingestion pipeline needs to be aware that an item is deprecated so that it does not ingest this item again - and Administrators are allowed to revert the status of such items to "APPROVED". Items once ingested or created but not meeting the quality criteria should be deprecated.

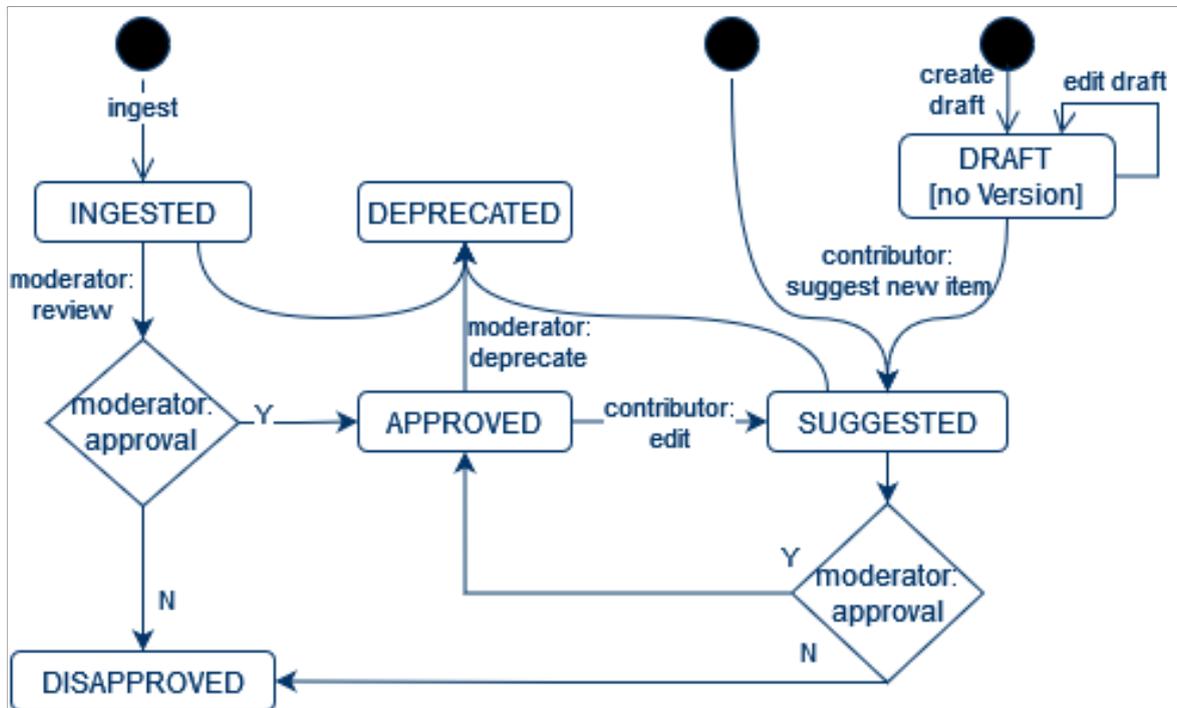


Fig. 2 - Transitions between statuses of items

Dynamic properties

Given the heterogeneity of the source data at hand, already the original data model has foreseen a mechanism for dynamic properties, i.e. a way to describe specific aspects of a resource without having to change the underlying data model implemented in the system. As a result, there are only a handful of dedicated attributes like label and description hard-wired into the data model, and most of the information about an MPEntity is being captured through a flexible set of Properties. This mechanism proved to be most instrumental in accommodating the diversity of information encountered in the different sources, specific to the different types of resources.

While this approach allows for maximum flexibility with respect to what can be captured, it hampers the traditional means of restricting or validating the input through constraints on the database level. This deficiency has been mostly remedied by introducing typing for the dynamic properties, which indicates if any given property is an integer, a string, etc. This same mechanism was employed to introduce so-called concept-based properties, i.e. properties which take the range of values from a controlled vocabulary. This is a crucial feature allowing for a flexible yet controlled categorisation of the resources. A dynamic system has been put in place where new properties can be defined as needed and can be associated with one or more vocabularies, which define the set of concepts or terms allowed as values for a given property.

For an overview of dynamic properties used in the SSH Open Marketplace, you can consult the editorial guidelines¹⁷.

Vocabularies

In many dynamic properties, the range of values needs to be restricted to a more or less fixed set of terms, or a controlled vocabulary. The data model foresees a generic mechanism, where such vocabularies can be imported into the system and attached as “allowed vocabularies” for individual dynamic properties. These constraints are checked by the system, i.e. only concepts from allowed vocabularies are valid inputs for given property.

The data model follows the SKOS schema¹⁸ and is capable of importing and exporting the vocabularies in SKOS. Expressing the terms used as concepts in vocabularies (in line with the SKOS model) allows the move from simple lexical terms to more complex semantic entries, with multiple lexical labels, definitions, examples etc., as well as semantic relations between these.

The vocabularies are exposed as autocomplete fields in the edit forms and as facets in the search interface. The additional information is made available on demand to the normal user in the search and detail view, thus providing the user with additional contextual semantic information (see following sections 4.3 and 4.5.1 for details on the user interface). More importantly however it is crucial for the curation work, which has to be based on well-defined and agreed upon terminology (see following section 3.2 for the role of vocabularies in fostering interoperability).

Typed relations

Another generic mechanism is the attribute *related* that allows defining relations between two *MPEntities*. The type of relation is not hard-wired in the data model but can be defined by the Administrator at runtime, so the allowed types of relations can be managed dynamically. A typical example of such a typed relation would be an *MPEntity* (e.g. a *Tool*) *isMentionedIn* another *MPEntity* (typically a *Publication*, or *TrainingMaterial*).

Similarly, the relation between an *MPEntity* and an *Actor* is kept generic, with the configuration of the applicable “roles” of an *Actor* with respect to an *MPEntity* left to Administrators at runtime. Typical roles include: *hasContributor*, *hasAuthor*, *hasFunder*.

Media*

While the need for media such as images has been recognized from the start, in practice it turned out that media objects are not well represented as simple dynamic properties, and need to be expressed as a separate class in the data model. Thus, *Media* has been added as a separate weak entity of *MPEntity*, allowing for 1:N relation and structured information about each media object. Next to a

¹⁷ SSH Open Marketplace Editorial Guidelines v.1 are published as annex to *D7.4 Marketplace Data population & curation*, and will be added to the “Contribute” pages of the SSH Open Marketplace website for its final release.

¹⁸ Simple Knowledge Organization System Reference: <https://www.w3.org/TR/skos-reference/> [11.11.2021]

dedicated field for caption, introduction of a type field for distinguishing between image, video, or object allowed for more elaborate handling of the media objects on the frontend side. Additionally an option was introduced to actually upload a file, alternatively to providing a source URL pointing to the media object.

externalIDs*

Based on the experience from mapping various sources, it was essential to introduce an array of qualified (external) identifiers both for *MPEntity* as well as *Actors*.

This is in acknowledgement of the fact that often, next to a dedicated homepage, there may be multiple equally authoritative representations of any given resource on the web, such as a git repository or a wikidata or wikipedia entry. This mechanism allowed these special URLs to be distinguished from other links, which may refer to various aspects of a given resource, such as documentation or helpdesk for a service, and to replace a number of URL-based dynamic properties, like *repository-url*, *wikidata-id*.

Provenance

MPEntity also features a few meta-attributes pertaining to the provenance of the information gathered about a given entity: *informationContributor* allows the User who entered the information about that entity to be tracked, *lastInfoUpdate* records the date and time of the last change to the entry.

3.2. SSH Open Marketplace data model in context

As presented in D7.1, the data model of the SSH Open Marketplace has been designed primarily based on the most important sources identified for data population, the kind of information expected by the users of the MP, and with the goal of having a flexible data model able to accommodate new kinds of information either coming from new sources or arising from the needs of discoverability. Nevertheless, three avenues have been pursued to foster interoperability:

- a) Extensive use of controlled vocabularies in concept-based properties for describing resources, employing existing external controlled vocabularies wherever possible (see Table 1 for an overview, and see the previous section on the implementation of vocabularies in the data model). All the vocabularies used in the Marketplace will be published via the Vocab service of DARIAH¹⁹ hosted by DARIAH/OEAW. External vocabularies are considered “closed”, i.e. no concepts are added or changed on the side of the Marketplace, as opposed to internal vocabularies based mostly on source data, which are considered “open”, i.e. they can evolve. Publishing the vocabularies and concepts as semantic artifacts in their own right will foster their reuse and thus interoperability between communicating systems. This holds for the Marketplace but also for the SSHOC family of services.²⁰

¹⁹ Vocab service: <https://vocab.dariah.eu/> [28.09.2021]

²⁰ This approach is in line with cross-WP activity led by CLARIN on harmonising vocabulary management and the corresponding infrastructure across the SSH cluster, called *SSH Vocabulary Commons*.

- b) Collecting and curating alternative and additional external identifiers (esp. from Wikidata, ORCID and other relevant reference resources) for the entities in MP (both items and actors), generating essentially identifier-sets that form the bridges to other LOD datasets. This also represents groundwork for potential enrichment from these sources, or even vice versa contributions of the curated information from MP to e.g. Wikidata.
- c) A mapping of the MP data model to the common SSHOC reference ontology²¹ has been proposed in collaboration with task 4.8 (led by FORTH). This mapping will be implemented so that the data in the Marketplace is available in RDF based on this ontology, exposed either through JSON-LD embedded metadata, an RDF-dump, or a triplestore with a SPARQL-endpoint.

property	vocabulary	type of vocabulary
geographical-availability	EOSC Geographical Availability List ²²	closed
life-cycle-status	EOSC Life Cycle Status List	closed
resource-category	EOSC Resource Category List	closed
intended-audience	audience	open
mode-of-use	Invocation type	open
language	ISO 639-3 Language Codes ²³	closed
keyword	Keywords from SSHOC MP	open
object-format	Media Types from IANA ²⁴	closed
discipline	ÖFOS 2012. Austrian Fields of Science and Technology Classification 2012 ²⁵	closed
license	Software License from SPDX ²⁶	closed
standard	SSK Standards List	open

²¹ See Bekiari, Chrysoula, Kritsotaki, Athina, & Tsouloucha, Eleni. (2020). SSHOC D4.18 SSHOC Reference Ontology (beta version) (v1.0). Zenodo. <https://doi.org/10.5281/zenodo.3744861> [12.10.2021]

²² EOSC related vocabularies are all available here:

<https://eosc-portal.eu/providers-documentation/eosc-provider-portal-resource-profile> [12.10.2021]

²³ ISO 639 Code Tables: https://iso639-3.sil.org/code_tables/639/data [12.10.2021]

²⁴ IANA Media Types: <https://www.iana.org/assignments/media-types/media-types.xhtml> [12.10.2021]

²⁵ ÖFOS 2012. Austrian Fields of Science and Technology Classification 2012: <https://vocabs.dariah.eu/oefos/en/> [12.10.2021]

²⁶ SPDX License List: <https://spdx.org/licenses/> [12.10.2021]

activity	TaDiRAH 2 ²⁷	closed
publication-type	The Bibliographic Ontology Concept Scheme ²⁸	closed

Table 1 - Vocabularies used in concept-based properties

4. Implementation

In this chapter, the actually implemented application is described, on the backdrop of the envisaged setup laid out in D7.1. The first section gives an overview of the individual components and how they evolved in the process of implementation, the following sections give details about the individual components and other aspects of the system - like its configuration, deployment or API.

4.1. From specification to implementation

Figure 3 depicts the system architecture as envisioned at the beginning of the project and formulated in *D7.1 System Specification*. Figure 4 is an update of the system architecture representing the state of the affairs post hoc, i.e. towards the end of the project, when the system has already been largely implemented. While the principal components of the system have remained as originally foreseen, a few refinements and adjustments to the original plan have occurred in the course of the implementation, stemming from experience gathered during the development as well as feedback from the involved parties (test users, developers, curation team).

It is to be noted that given the numerous components involved in the final architecture, a very strict regime has been implemented with respect to access to the data, in order to ensure data consistency and encapsulation, in that the only way to read or edit the primary data stored by the core component is via the REST-API. Both the primary frontend application, as well as the ingestion pipelines and curation scripts interact with the data by means of the same well-defined API.

²⁷ TaDiRAH: Taxonomy of Digital Research Activities in the Humanities: <https://vocabs.dariah.eu/tadirah/en/> [12.10.2021]

²⁸ Bibliographic Ontology BIBO: <https://github.com/structuredynamics/Bibliographic-Ontology-BIBO> [12.10.2021]

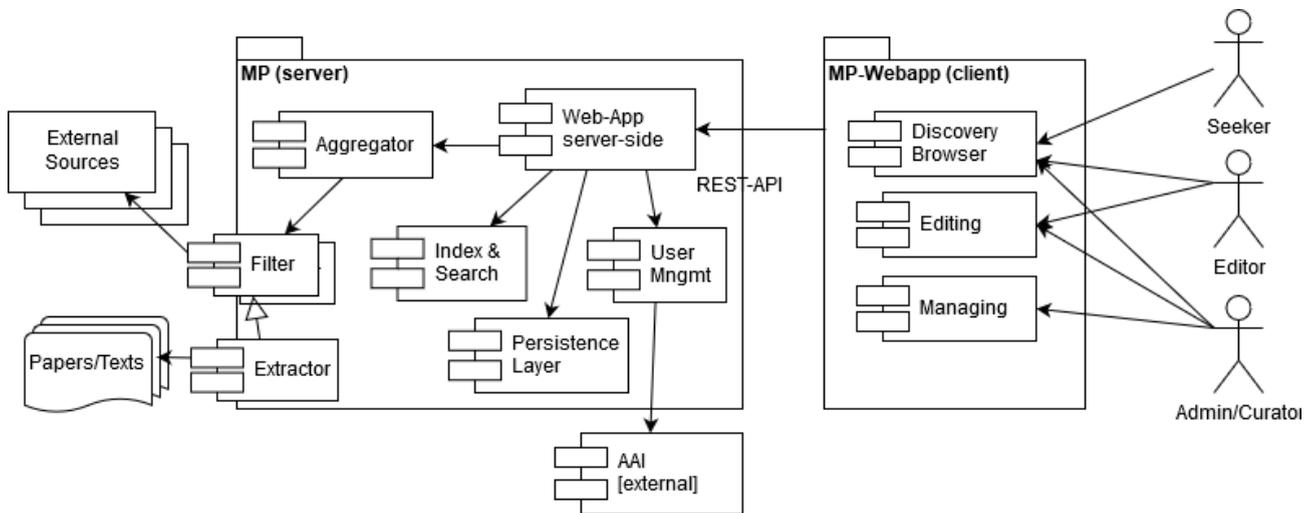


Fig. 3 - System Architecture Diagram as introduced in D7.1 System Specification

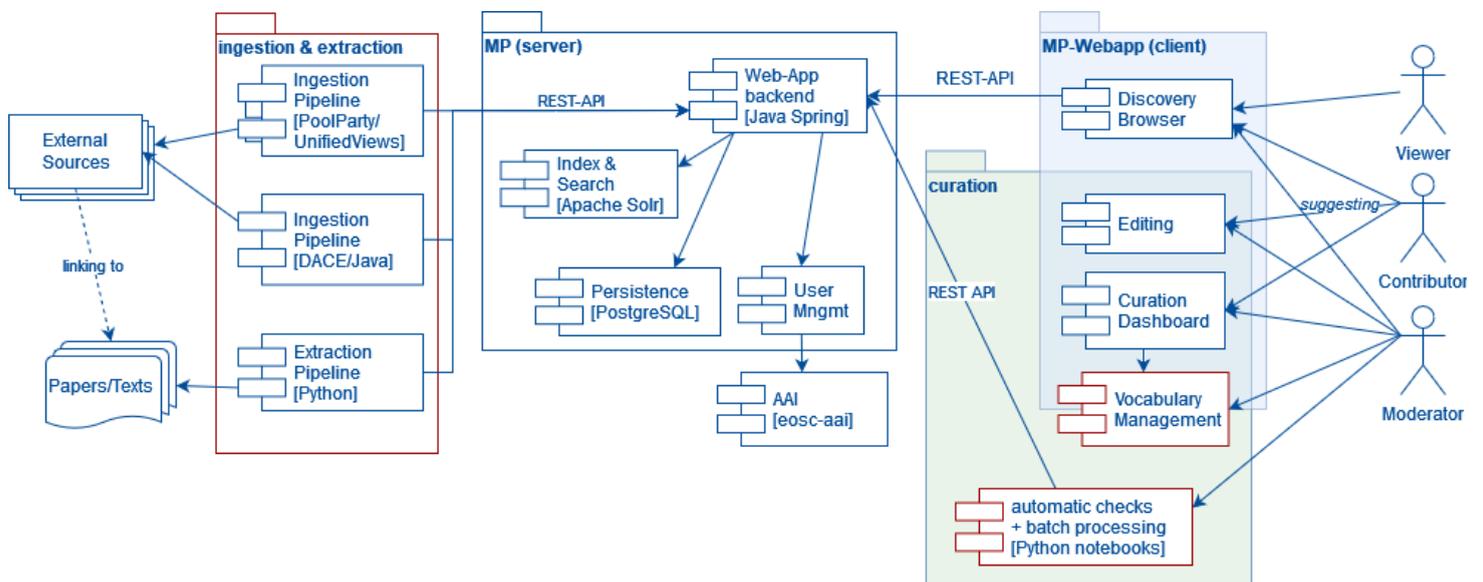


Fig. 4 - Update of the system architecture (major changes to original plan are marked red)

Table 2 lists the individual components as originally envisaged in the system specification, contrasting them with the actual implementation in the final system.

Components	Specification	Implementation
Server-side web application	Comprises primary persistence, business logic and exposes REST API, including methods used by the frontend application.	Server application implemented in Java using the Spring framework. ²⁹
- Persistence Layer	Stores information offered by and necessary to operate the Marketplace, e.g. information about users, metadata about entities available in the platform.	PostgreSQL ³⁰ database implementing the data model allows storage of all the information about the MP items, as well as all auxiliary entities, like actors, or media, and all previous versions of the items.
- Index & search	Provides efficient mechanisms (including a search engine) supporting discovery features of the platform, keyword and faceted search.	An Apache Solr component. Only available to the main Java application, which passes on search requests transparently to the solr index and returns back results via corresponding REST endpoints.
- Aggregator	The component is responsible for automatically harvesting information from identified sources, transforming and ingesting it into the platform. It requires custom filters, mapping and a clear policy on how to deal with updates/conflicts.	Implemented as separate components, independent of the core application, communicating with it solely via the REST API. Originally implemented using PoolParty Unified Views, meanwhile a new system DACE has been developed. See the <i>Ingestion</i> section below for details.
- Extractor (optional)	Can be considered as a special kind of Aggregator that takes text as input and tries to extract relevant information	Given its experimental nature and complex processing/specific requirements (use of NLP methods), this

²⁹ Spring Framework is an application framework and inversion of control container for the Java platform

³⁰ Relational database management system (RDBMS) used. See <https://www.postgresql.org/> [24.09.2021]

	<p>from it, then again ingesting it into the platform.</p> <p><i>This is considered as an optional component serving an experimental aspect of the platform.</i></p>	<p>component has been implemented independently of the set of ingestion pipelines using Python.</p> <p>See the <i>Extraction module</i> section below for details.</p>
- User management	<p>Primarily we aim to rely on Identity Federations (Shibboleth, OpenID). Nevertheless, the system needs to have a “local” representation of the user, and also we need a fall-back to register locally if all else fails.</p> <p>This component also comprises a user profile that could capture the user’s search history or allow her to bookmark certain items or store queries, etc.</p>	<p>The system implements a hybrid user management allowing both local and remote users, remote ones authenticated by means of federated identity relying on the EOSC AAI.</p> <p>See the <i>EOSC AAI</i> section below for more details.</p> <p>Personalisation features (user’s search history etc.) haven’t been implemented.</p>
Rich client application with modules:	<p>primary user interface to explore the data offering searching and browsing capabilities.</p>	<p>Implemented using React framework, communicating with the server solely through the defined REST-API.</p>
- Search/Discovery	<p>Faceted browser and full-text search for the end-user to explore the content.</p>	<p>Both modules form the integral part of the client application.</p> <p>See the <i>Frontend component</i> section for details.</p>
- Detail View	<p>A detailed view of each entity, gathering and presenting all existing contextual information, also allowing to navigate “sideways” to similar entities based on the contextualisation.</p>	
- (Micro-)Editing	<p>Collaborative editing/curation of the information/content.</p> <p>The authoring mode, allowing to make changes to data.</p> <p>Only available to logged-in users, distinguishing roles. General logged in users can suggest changes which need to be approved by a Moderator.</p> <p>Similar to wiki-data, the idea of micro-editing is that a user can suggest</p>	<p>Edit forms implemented as part of the main client application. Users with appropriate permissions can switch from view-mode to edit-mode on any item, minimizing the effort to propose changes to the existing information.</p> <p>Micro-editing has not been implemented, due to both technical difficulties and usability concerns - a clear distinction between view and edit</p>

	just one specific fact or a piece of information for an existing entity entry in a quick and intuitive fashion.	mode seemed more suitable. See the <i>Frontend component</i> section for details.
- Managing	A module for the Moderators, power-users, a dashboard informing about status and history of automatic imports (aggregation) and checks as well as manual (suggested) changes (moderation of the community).	<p>The curation dashboard is an integral part of the frontend application, available to users with appropriate rights (Moderators, Contributors), it gives an overview of the status of the data in the system with respect to its curation status.</p> <p>Automatic checks accompanied by corresponding batch processing methods have been implemented as a set of python notebooks to be operated by the moderators probing the data via REST API exposed by the backend, subsequently performing a number of statistics and data quality checks. See the <i>Curation Components</i> section for details.</p>
- Vocabulary Management	A big part of the Managing and curation will be dealing with / curating vocabularies. This would justify a dedicated (potentially external) tool for managing the vocabularies.	Originally relying on the vocabulary management capabilities of the PoolParty suite/solution. Meanwhile MP-API allows for a basic management of vocabularies and concepts. See the <i>Vocabulary Management</i> section for details.
Data Lab / Notebook (optional auxiliary service)	Not considered an integral part of the Marketplace, rather expected to be an externally provided but tightly connected service that allows capturing and “replaying” recipes, workflows. Motivation:	Not implemented, however the Data Lab technology (Python notebook) has been employed to implement the automatic curation procedures.

	Describing workflows in “data notebooks” (e.g. ipynb – the python notebook), which combine prose and code and can be inspected and edited through a browser and executed server-side is becoming popular very fast. This seems to be an ideal means to accompany the solutions in the Marketplace, with executable code (where possible).	
--	---	--

Table 2 - Components of the system architecture: foreseen vs. implemented

4.2. Backend component

The core part of the backend component is implemented in the Java Spring framework. A PostgreSQL database serves as the primary persistence layer, Apache Solr is used for additional indexing to allow for fast searching and faceting of the data. Both the database and the Solr instances are internal to the system; the only way to access the data by other components or third-party applications is via a REST API exposed by the backend component. This API offers the necessary endpoints to retrieve or manipulate any of the entities managed by the system. The endpoints, the available parameters, as well as the response formats are described using OpenAPI³¹. API endpoints can be divided into six categories in line with the main entities in the data model:

- CRUD³² and Search endpoints of items offered by SSH Open Marketplace
- CRUD and Search endpoints of Property types and Vocabularies used to describe these items
- CRUD and Search endpoints of Actors and their roles in the items
- CRUD endpoints for Sources from which the items come
- Endpoints for uploading/importing media and connecting them to the items
- Endpoints for signing-in, both with local accounts registered in the backend component and via external identity providers (IdPs) available by OAuth2³³ protocol in EOSC. See section *EOSC AAI* in the next chapter for more details.

The backend component preserves the whole history of items (tools and services, datasets, training materials, publications, and workflows) and manages their statuses. These statuses depend on the called method creating subsequent versions of items (create/update/delete) but also on the role of the

³¹ OpenAPI specification: <https://swagger.io/specification/> [11.11.2021]

³² CRUD stands for the generic operations: Create, Read, Update, Delete

³³ OAuth: <https://oauth.net/> [18.10.2021]

user who calls the method (Contributor vs. Moderator). Accepted versions are provided via read and search methods for unauthenticated users. Authenticated users are able to retrieve other versions depending on their roles and whether they are creators of a particular version. The backend component uses JWT tokens³⁴ for user authentication.

To ensure stability of the API, more than 400 integration tests were implemented as a part of the project, allowing to automatically test all REST API endpoints. See section 4.7 below for more information regarding deployment and configuration of the system.

4.3. Frontend component

Based on the user requirements and the system specification presented in D7.1, some low-fidelity sketches were designed by DARIAH/PSNC in September 2019 (see Fig. 5). After a round of feedback from all involved partners, detailed mockups of the main types of pages - home page, search result and item detail view (see Fig. 6) - have been devised and discussed with future users allowing the frontend developer from DARIAH/OEAW to start the implementation of the client application - towards the Minimal Viable Product in December 2019.

The frontend is implemented as a Next.js application, and utilises React Query for client-side caching. As accessibility of the application should be ensured for all audiences, UI components closely follow WAI-ARIA Authoring Practices 1.2³⁵.

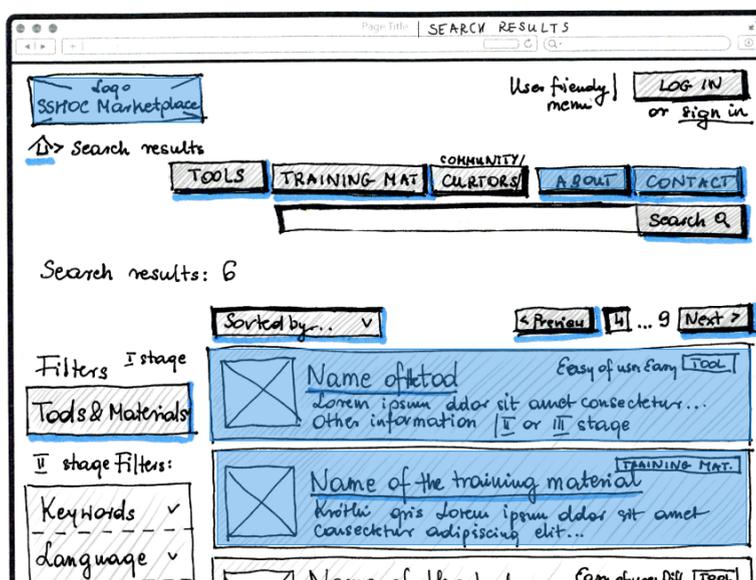


Fig. 5 - low-fidelity sketches of the search result page

³⁴ JSON Web Token: https://en.wikipedia.org/wiki/JSON_Web_Token [11.11.2021]

³⁵ WAI-ARIA Authoring Practices 1.2: <https://www.w3.org/TR/wai-aria-practices-1.2/> [13.10.2021]

Furthermore, to ensure user-friendly management of the SSH Open Marketplace, a basic Content Management System (CMS) to edit the content of the static pages of the SSH Open Marketplace has been set up. This interface allows Moderators of the Marketplace to edit, in Markdown³⁶, two main menus of the portal: the “About” and “Contribute” pages hosting and structuring the user’s documentation.

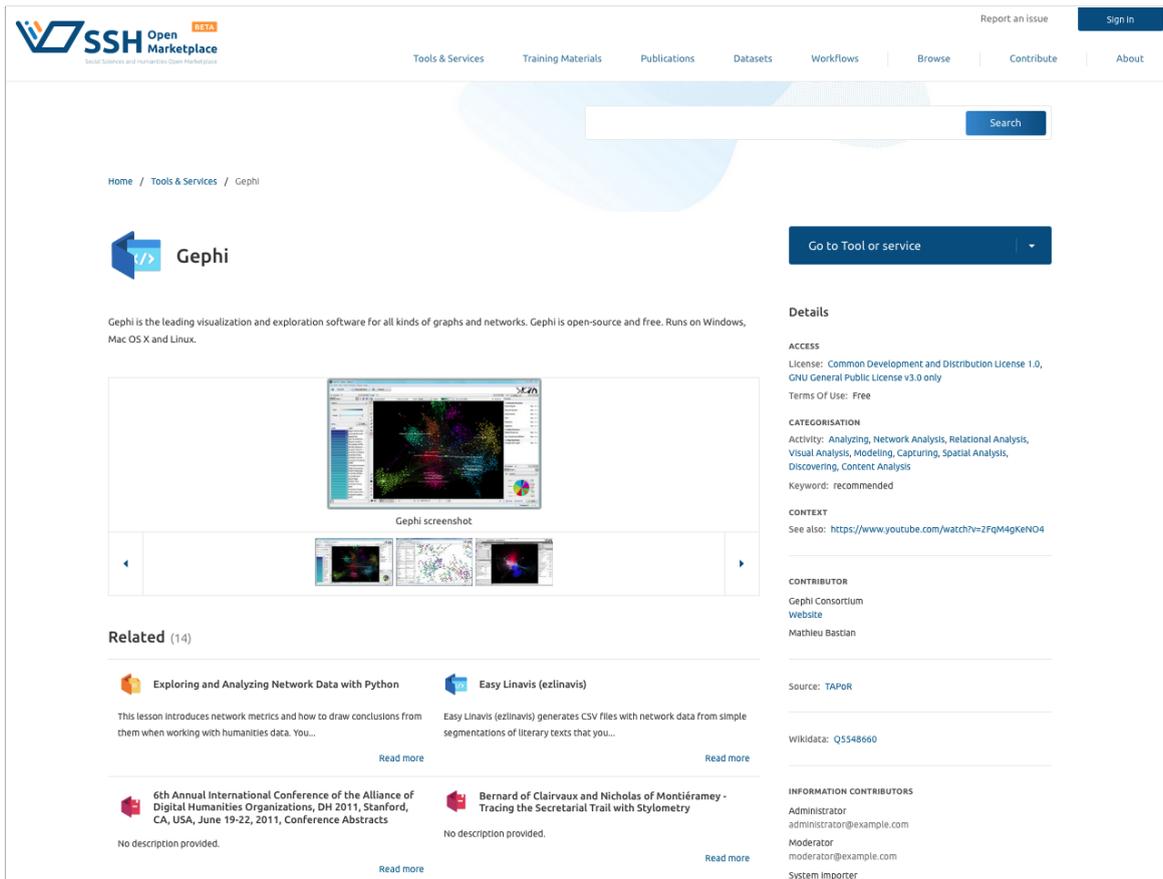


Fig. 6 - Screenshot of the Gephi tool - Item detail view

Finally, the edit forms and an editorial dashboard - see *Curation Components* section for more details - have been implemented and are the main user interfaces for data curation of the Marketplace.

4.4. Ingestion

A critical part of a discovery service like the SSH Open Marketplace, assuming a role of an aggregator, is a component for collecting and ingesting information from external sources. In the original system specification, this functionality was encapsulated in the aggregator component, foreseen as part of the

³⁶ Markdown: <https://en.wikipedia.org/wiki/Markdown> [11.11.2021]

core server-side application. In the final system, it has been implemented as a separate component which communicates with the core application via the REST API.

Different ingestion pipelines have been employed over the course of the project, two of them are described in more detail in the next two subsections. Originally, PoolParty Unified Views was used, as it was provided by a project member - SWC (see subsection 4.4.2). Even though PoolParty is a comprehensive well-established solution, its licensing terms after the end of the project prompted a discussion in the WP7 team aimed at finding an alternative solution for the ingestion task. To this end, a set of features for an ingestion component has been compiled and the following aggregator solutions frameworks have been assessed against it:

- Leopoldina, a regional aggregator for Europeana developed by PSNC;
- a combination of the METIS framework³⁷ and ECloud;
- the DNET approach from OpenAire;
- the respective aggregators of Archives Portal Europe and the Jewish Heritage Network;
- the PoolParty UnifiedViews used so far in the SSHOC context, and
- the MoRe aggregator³⁸.

In parallel, a discussion with the technical team of the TRIPLE project³⁹ has also been conducted, to understand if the AirFlow framework that the discovery platform GoTriple decided to rely on could be an option for the SSH Open Marketplace as well⁴⁰.

After these investigations, it turned out that none of the solutions (besides PoolParty) can be used without further adaptation, the decisive aspect became the availability and capacity of a software development team able to adjust any of the solutions to the specific needs of SSH Open Marketplace. This gave way to the decision to further develop Leopoldina by DARIAH/PSNC resulting in the thoroughly refactored system Data Aggregation and proCessing Engine (DACE) further described in section 4.4.3.

The list of specific sources to harvest during the development phase of the SSH Open Marketplace has been managed in task 7.3. The procedures for compiling the list, as well as the list itself, are available under *D7.3 Marketplace Interoperability*.

³⁷ Metis framework: <https://github.com/europeana/metis-framework> [28.09.2021]

³⁸ MoRe aggregator: <http://more.dcu.gr/?p=home> [28.09.2021]

³⁹ TRIPLE project website: <https://project.gotriple.eu/> [28.09.2021]

⁴⁰ See the "Exploring the SSH data landscape: thematic discovery portals in the EOSC" session of the Realising the EOSC event for more details: Frank Fischer, Yoann Moranville, Laure Barbot, Laurent Capelli, Virginie Ngo, Suzanne Dumouchel, Emilie Blotière, Matej Ďurčo, Dieter Van Uytvanck, Alexander König, Arnaud Gingold, Aleksandra Nowak, & Tomasz Parkoła. (2020, November 25). Exploring the SSH data landscape: thematic discovery portals in the EOSC. REALISING THE EUROPEAN OPEN SCIENCE CLOUD Towards a FAIR research data landscape for the social sciences, humanities and beyond. Zenodo. <https://doi.org/10.5281/zenodo.4290599> [28.09.2021]

4.4.1. Ingestion workflow

The ingestion workflow supports the mass data population pipeline of the SSH Open Marketplace. Figures 7 and 8 below illustrate the workflow itself and the different components involved in the process.

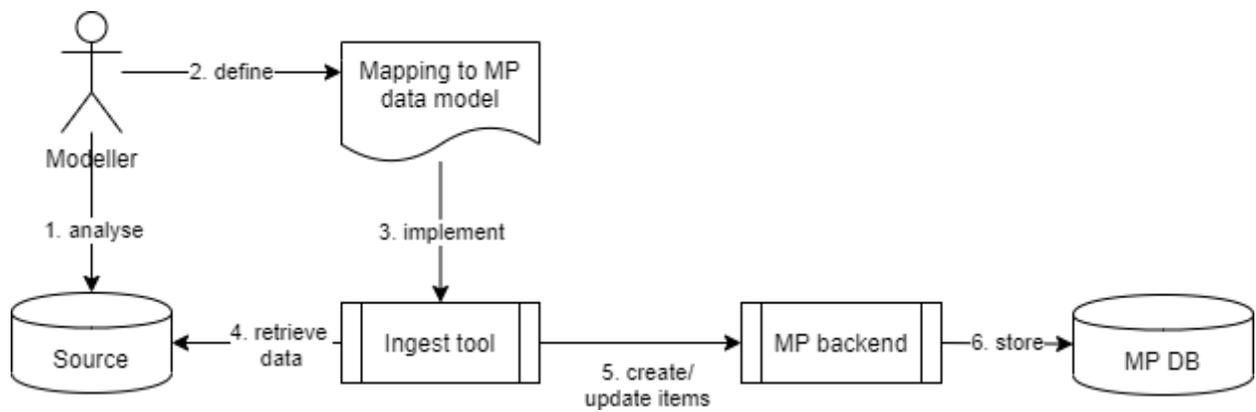


Fig. 7 - Overview on the different components of the ingestion workflow

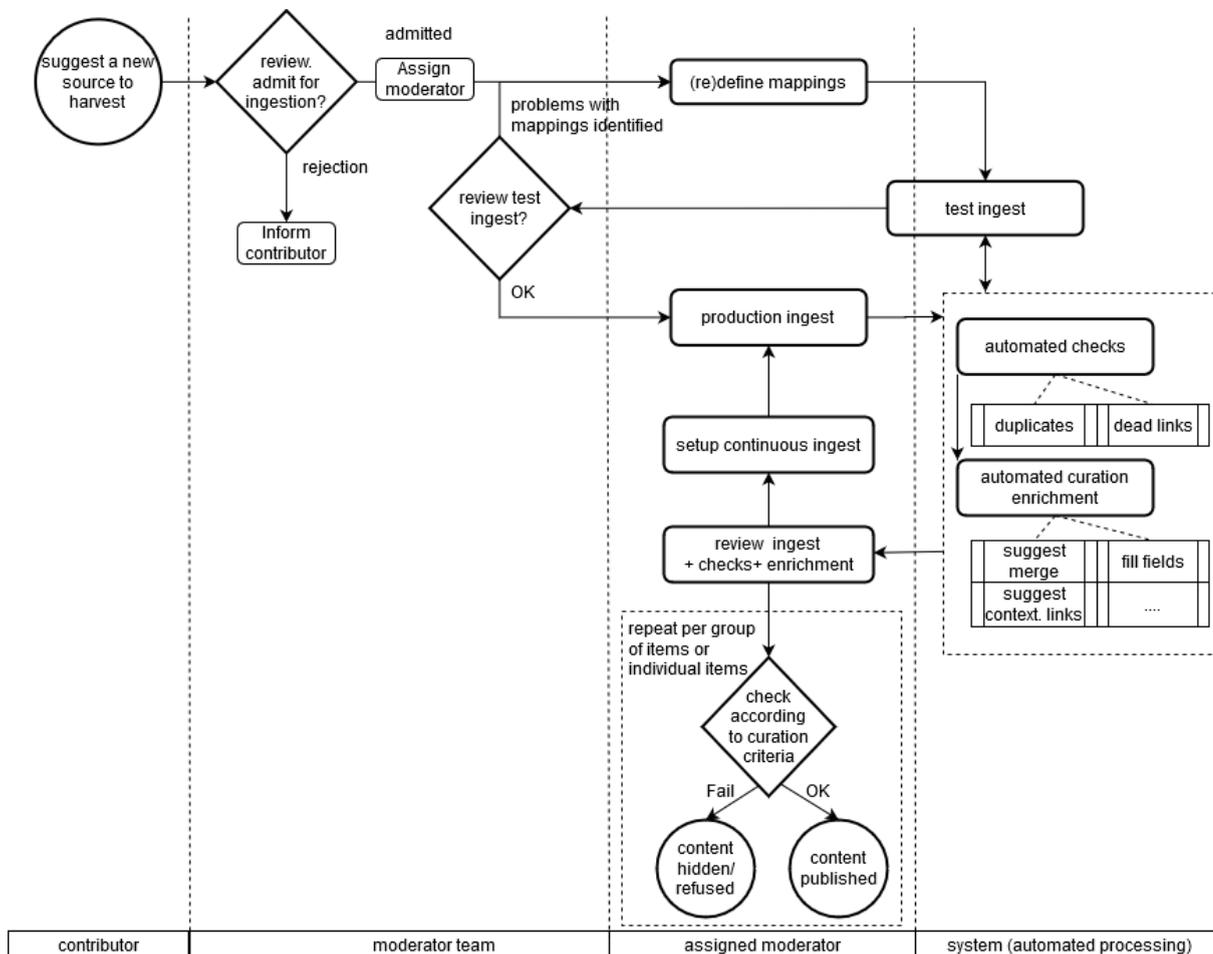


Fig. 8 - Ingestion workflow

This ingestion workflow can be summarised by the following steps.

1. **Suggest a new source to harvest.** A source can be suggested by a Contributor or Moderator via a contact form. The decision to include a new source has to be met on a consensual basis within the Moderators' team. *D7.3 Marketplace - Interoperability* includes a list of criteria that the Editorial Team of the Marketplace should take into consideration for future sources to harvest.
2. **Define mappings.** A mapping from the source data model to the Marketplace data model has to be devised in each case. This mapping is prepared in a spreadsheet by one of the Moderators and reviewed by at least another. This mapping represents the prescription for the custom ingestion pipeline.
3. **Implement a custom pipeline.** Based on the mapping defined in the previous step, a custom ETL (extract, transform, load) pipeline is implemented, which fetches the data from the source, iteratively processes all the items and maps the structured data in the source format to the MP data model, ingesting the transformed items expressed as JSON-objects via the API of the Marketplace.

4. **Test ingest and review.** After the mappings are defined, ingestion against a test instance and automatic checks are performed, as the basis for a review of the quality of the mapping. Identified problems inform refining of the mappings in an iterative process, until the test ingest passes the review (by the Moderators team). This work will be handled by the assigned Moderator, if necessary in cooperation with the Administrators.
Note: in this stage, the checks are performed, but no curation or editing of the data is performed, only adjustments to the mapping.
5. **Production ingest.** After the test ingest has been approved, the assigned Moderator can invoke the ingestion against the production instance. The ingest triggers automatic checks delivering reports for manual curation via the curation dashboard. These steps are detailed in *D7.4 Marketplace - Data population & curation*.
6. **Continuous ingest.** For sources, the content of which is expected to change substantially over time, the production ingest is configured as a recurring task. The ingestion pipeline is able to match the source items against Marketplace items and updates them accordingly, and processes previously unseen items to create new Marketplace items.

4.4.2. Pool Party UnifiedViews Pipeline

UnifiedViews (UV) is an ETL-tool for RDF data. The Data Processing Units (DPUs) provide functions that enable users to cover the extraction and data transformation tasks they need in order to prepare for and use data with PoolParty.

The DPUs can be chained into a pipeline, output from one DPU serving as input for another DPU, allowing the composition of complex processing pipelines out of simple functions encapsulated in individual DPUs. At the core of the processing is a RML Schema mapping DPU, which can execute mappings from various sources in JSON, XML or CSV format into an RDF data model using the RML specification⁴¹. The advantage of using RML for the definition of mappings is that the mappings are interoperable and are considered as cross-platform, since the mappings can be used in a variety of platforms (Java, python, Scala, JS). One example of the mapping definition can be seen in the SSHOC data-ingestion repository⁴². In this example, JSONPath expressions⁴³ are used to define the mappings of specific fields on each JSON file to the respective RDF properties.

UV has been used as a tool to ingest metadata from various sources into the SSH Open Marketplace⁴⁴. For every distinct source, three different pipelines exist: data acquisition; Users ingestion; and

⁴¹ RML specification: <https://rml.io/specs/rml/> [28.09.2021]

⁴² Example of the Programming Historian actors mapping definition: <https://gitlab.gwdg.de/sshoc/data-ingestion/-/blob/master/repositories/programminghistorian/users/mapping.ttl> [28.09.2021]

⁴³ JSONPath - XPath for JSON: <https://goessner.net/articles/JsonPath/> [11.11.2021]

⁴⁴ The SSHOC UV instance is available (with credentials) at: <https://sshoc-unifiedviews.poolparty.biz/> [24.09.2021]

Marketplace Entity Ingestion. This design decision was made to reduce used resources, and have better control of the steps. Pipelines are named using the name of the source and the phase of the ingestion.

1. **Source - Data acquisition.** This is where metadata are extracted from the source. The easiest case is when metadata exist in some repository (i.e. github) as JSON, XML, YAML or CSV. This DPU takes as configuration a URL, that executes a download from some location (usually a github repository).

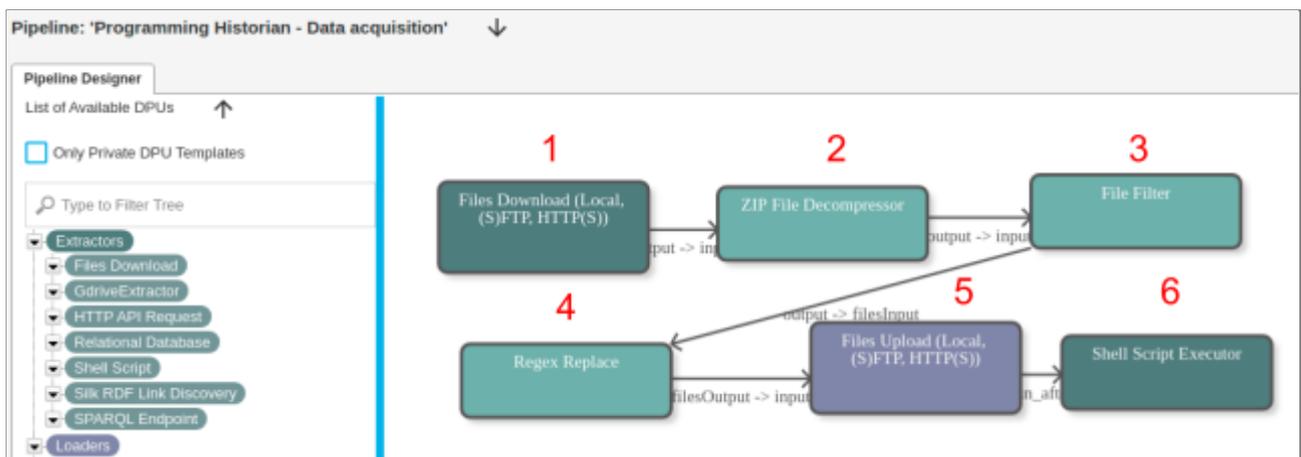


Fig. 9 - Screenshot of the Data Acquisition DPUs in Pool Party

2. **Source - Ingesting users.** *User instances* have to exist in the platform before any other resource ingestion, prompting this a separate step.
3. **Source - lessons|tool|scenarios|steps ingestion.** This is where the mappings of the source data to the Marketplace data model are created. These pipelines can be used as templates for other sources in the respective *MPEntity* resource.

Entity ingestion pipelines are more complex than the user pipelines. Tasks like annotation with vocabularies, cleaning and transformation are handled either through the RML mappings, or SPARQL queries.

4.4.3. DACE

DACE⁴⁵ is an Open Source aggregation framework that has been developed by DARIAH/PSNC following the SSH Open Marketplace needs and requirements in terms of aggregation and processing. DACE consists of several applications (microservices) that can be used as a set to build a harvesting,

⁴⁵ See the Data Aggregation and proCessing Engine (DACE) repository: <https://gitlab.pcoss.pl/dl-team/aggregation/dace> [13.10.2021]

processing and ingesting workflow. Applications communicate with each other using asynchronous messaging. The framework is flexible and the deployment does not have to consist of all the available applications but only those which are really needed for the specific processing workflow. Each application can be scaled and run in multiple instances if needed.

DACE applications are divided into two main parts⁴⁶:

1. **harvesting applications**: their goal is to communicate with the source, run a specific query/request, parse the response and save the results in the DACE database. There are components supporting standard protocols (like OAI-PMH⁴⁷) and others built to communicate with custom data sources (e.g. HTTP interface listing records in a JSON file). The latter are used in SSHOC.
2. **processing applications** used for transforming source records into the desired form, so they can be saved into the target database (SSH Open Marketplace, in this case). DACE deployment in SSHOC uses a dedicated processing application. It performs a mapping step - JSON to JSON transformation - using the JOLT library. For each data source a separate mapping is prepared. Each processing application also executes an enrichment step, linking the result of the mapping to Marketplace entities (e.g. for sources, actors) or concept entities (from vocabularies) and a final indexing step in which the record is created or updated in the Marketplace instance using its REST API.

⁴⁶ Detailed DACE user documentation is available at: <https://gitlab.pcass.pl/dl-team/aggregation/dace/-/wikis/home> [24.09.2021]

⁴⁷ See Open Archives Initiative - Protocol for Metadata Harvesting: <https://www.openarchives.org/pmh/> [12.10.2021]

Jolt Transform Demo Using v0.1.1

Here you can experiment with the stock Jolt Transforms without having to download and run the Java code.

Json Input	Jolt Spec	Output / Errors
<pre> 1 [{"id": "dariah-de-collection-registry-tutorial", 2 "title": "DARIAH-DE Collection, Registry Tutorial", 3 "lang": "en", 4 "date": "2021-06-23T00:00:00.000Z", 5 "version": "1.0.0", 6 "authors": [7 { 8 "id": "ola-nowak", 9 "avatar": "/assets/images/cms/people/dariah/ola-nowak", 10 "firstName": "Ola", 11 "lastName": "Nowak", 12 "twitter": "SomeTwitterId", 13 "orcid": "????-0002-5350-067X" 14 }, 15 { 16 "id": "sara-james", 17 "avatar": "/assets/images/cms/people/dariah/sara-james", 18 "firstName": "Sara", 19 "lastName": "James", 20 "orcid": "6666-0002-5350-067X" 21 } 22], 23 "tags": [24 { 25 "id": "research-infrastructures", 26 "name": "Research infrastructures", 27 "description": "Research Infrastructures" 28 }, 29 { 30 "id": "open-science", 31 "name": "Open science" 32 } 33] 34 } </pre>	<pre> 1 [{"operation": "modify-default-beta", 2 "spec": { 3 "authors contributors editors": { 4 "*": { 5 "fullName": "=concat(@1,firstName)", 6 "type": { 7 "code": "license" 8 } 9 }, 10 "remote": { 11 "date": "null", 12 "url": "=concat('source-item-id:',@2, 13 'remote-id:',@3)" 14 } 15 } 16 }, 17 { 18 "operation": "shift", 19 "spec": { 20 "uuid": "TEMP.sourceItemId", 21 "title": "TEMP.label", 22 "abstract": "TEMP.description", 23 "remote": { 24 "url": "TEMP.accessibleAt[]", 25 "date": "TEMP.TEMP_DATE.dateCreated" 26 }, 27 "date": "TEMP.TEMP_DATE.date", 28 "targetGroup": { 29 "Domain researchers": { 30 "#researcher": "TEMP.TEMP_PROPERTIES.#researcher", 31 "#audience": "TEMP.TEMP_PROPERTIES.audience" 32 } 33 } 34 } 35 } </pre>	<pre> 1 [{"sourceItemId": "S9eZzEjhtGfPjYQ_V9Vze", 2 "label": "DARIAH-DE Collection, Registry Tutorial", 3 "description": "This tutorial explains the full range of Dariah services and how to use them.", 4 "accessibleAt": ["https://dfa.de.dariah.eu/collections/ola-nowak"], 5 "properties": [{ 6 "type": { 7 "code": "license" 8 }, 9 { 10 "concept": { 11 "uri": "https://creativecommons.org/licenses/by/4.0/" 12 } 13 }, { 14 "type": { 15 "code": "keyword" 16 }, 17 "concept": { 18 "uri": "Research infrastructures" 19 } 20 }, { 21 "type": { 22 "code": "keyword" 23 }, 24 "concept": { 25 "uri": "Open science" 26 } 27 }, { 28 "type": { 29 "code": "keyword" 30 }, 31 "concept": { 32 "uri": "Open access" 33 } 34 } 35] 36 } </pre>

Fig. 10 - JSON to JSON transformation on the JOLT demo site for DARIAH-Campus source

4.5. Curation Components

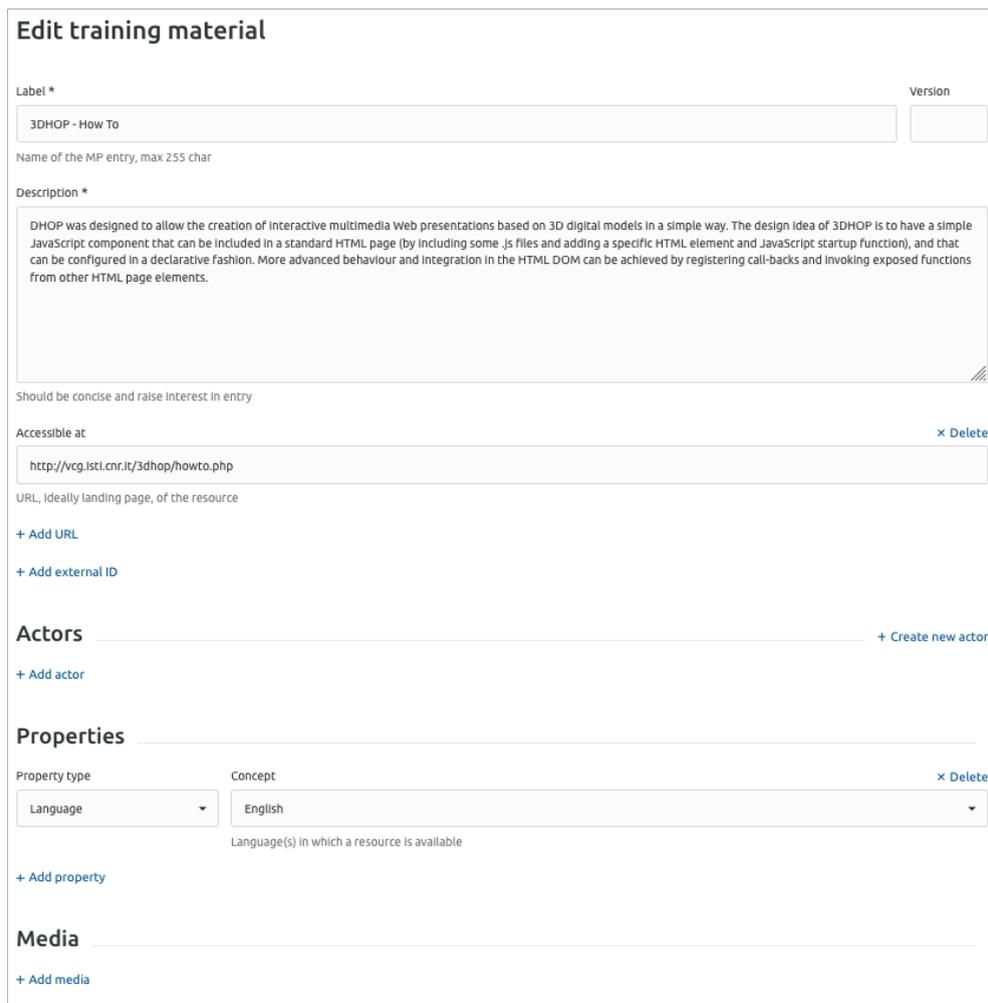
This section subsumes components of the overall system relevant to the critical task of ongoing curation of the data contained in the Marketplace. The original idea to combine automatic checks with the possibility to intervene manually has manifested itself in the following setup:

- **Item Edit Form** (see 4.5.1) - each item can be edited by users with sufficient permissions, the edit form being tightly integrated to the web application, allowing switching between view and edit mode.
- **Curation Notebooks** (see 4.5.2) - in order to maximize flexibility and expressivity, Jupyter Python Notebooks have been introduced as a means to systematically analyse the dataset according to the editorial guidelines and agreed upon quality checks. Implementing the automatic checks as a separate component independent of the core application was a deliberate design decision, to ensure a) a stable core application, b) simple extension of the checking procedure (without a need for changes to the code of the core application), c) separation of concerns (central data management, vs. curation tasks).
- **Curation Dashboard** (see 4.5.3) - a module of the web application component available to users with elevated permissions (Moderators), providing an overview of the status of the items in the Marketplace with focus on the quality of the information and the ensuing curation tasks.

4.5.1. Item Edit Form

This basic form allows manual creation and/or editing of all information about a specific item and supports three modes of editing: draft, suggestion and edit. For suggestions and edits, changes are recorded as a new version of the entry, however a suggestion is flagged as “suggested” and thus is not visible publicly, until reviewed and approved by the Moderator. To give a complete overview and support the curation workflow, the edit form features a version history, visible to Moderators, listing all previous versions of an item.

A major challenge in implementing the edit forms is to reflect the dynamic and flexible data model, while at the same time allowing for customisation depending on the item type. To this end, the frontend application implements a number of custom widgets/web-components to support the data authoring, such as autocomplete fields for concept-based properties, as well as actors and related items.



Edit training material

Label * Version

3DHOP - How To []

Name of the MP entry, max 255 char

Description *

DHOP was designed to allow the creation of Interactive multimedia Web presentations based on 3D digital models in a simple way. The design idea of 3DHOP is to have a simple JavaScript component that can be included in a standard HTML page (by including some .js files and adding a specific HTML element and JavaScript startup function), and that can be configured in a declarative fashion. More advanced behaviour and integration in the HTML DOM can be achieved by registering call-backs and invoking exposed functions from other HTML page elements.

Should be concise and raise interest in entry

Accessible at x Delete

http://vcg.lstl.cnr.it/3dhop/howto.php

URL, Ideally landing page, of the resource

+ Add URL

+ Add external ID

Actors + Create new actor

+ Add actor

Properties

Property type Concept x Delete

Language English

Language(s) in which a resource is available

+ Add property

Media

+ Add media

Fig. 11 - Screenshot of the edit form - “3DHOP - How To” item

4.5.2. Curation Notebooks

Right from the start, the curation process was foreseen as a combination of *manual* AND *automatic* tasks which inform each other. Besides the question of the specific checks that can be automated, there was the question of the general mechanism for implementing such automatic checks. The following major requirements have been identified:

1. These checks should be flexible, i.e. they shouldn't be hard-coded as part of the backend, so that rebuilding and redeployment of the whole backend would be required for every change in the checks. Rather they should be considered a separate "component".
2. At the same time, the checks need to be systematic and the results of the checks need to be an integral part of the MP-data, so that the Moderators can operate on a consistent information set.

In response to this, a set of Jupyter Python notebooks has been developed by CNR that uses the MP API and readily covers the first point (flexibility). To satisfy the second point (systematic integration), this approach needed to be extended to write via the API, recording results of the automated checks to the database, or even introduce automated changes to the data. To this end, a set of curation properties has been introduced, which constitute the contract between the automatic and manual checks, passing on the information gathered in the automatic process to the Moderators. The curation properties are employed in the curation dashboard, allowing the Moderators to easily navigate through this procedural/curation information in combination with other dimensions of the data.

Given that the scripts interact with the system through the API, the access is subject to the same global authentication and authorisation rules as it is with the client application, i.e. only authorized personnel are able to perform write-operations via the API. Though the scripts can be easily adapted to run regularly (e.g. via cron, a basic server-side scheduling mechanism), it seems advisable at least in the initial period to run the scripts in a supervised mode, given that their outcomes provide the input for the human task of evaluating the results and conducting appropriate curation actions.

Although the notebooks are not fully finalised at the time of writing this report, the following general structure is foreseen and to large extent already implemented:

- **"utils" library** - implements the methods used to perform analysis and checks in the notebooks.
- **ingest overview** - a notebook to review the sources ingestion presents some statistics about the number of values per attributes and properties to easily compare the ingestion and the mapping phases.
- **global data overview** - a notebook presenting a global overview of the Marketplace data with descriptive statistics of:
 - the item type

- the provenance (by sources or by users)
- the metadata coverage rate (using the number of null values in mandatory and recommended fields per item type)
- the quality of the description field based on its length
- the contextualisation quality that presents an overview of the number of relations per items
- **type-based checks** - a set of notebooks with specialized checks for the main types of entities. These notebooks also feature/invoke methods writing results of the checks back to the system via the curation flags properties.

The following are some of the checks implemented as Python methods, invocable within the curation notebooks (see D7.4 for a more detailed discussion)⁴⁸:

- checking accessibility of links / dead links
- missing/null values, coverage of fields
- description length
- duplicate items and duplicate actors

4.5.3. Editorial Dashboard

This component provides a dedicated user interface to overview and manage items, actors and users. Coming from inspirations such as the CLARIN Harvest Viewer⁴⁹, the editorial dashboard provides an overview for Moderators of the sources’ records being retrieved by the Marketplace. This overview varies depending on the user roles. It allows Contributors to see the list of their draft and suggested items, Moderators to manage the items to moderate, the sources labels and the actors. In addition to these actions, Administrators can manage the user roles.

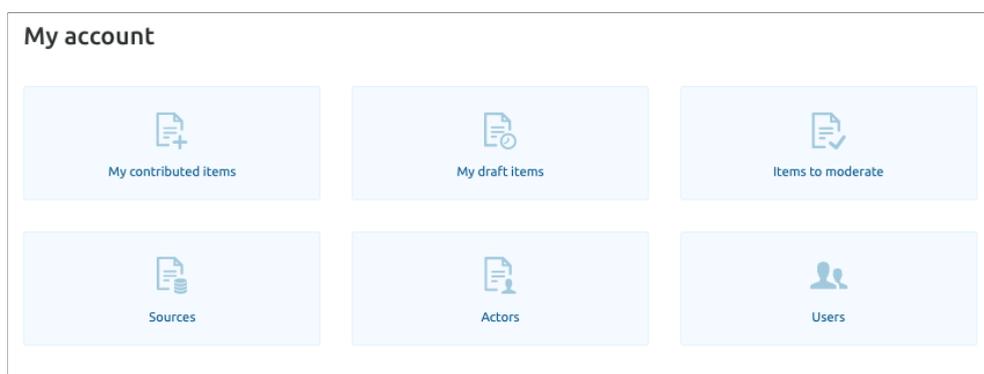


Fig. 12 - screenshot of the “My account” homepage for Administrator

⁴⁸ Potential further checks are being considered like consolidation of vocabularies or finding and suggesting related items for example.

⁴⁹ See the CLARIN Harvest Viewer: <https://vlo.clarin.eu/oai-harvest-viewer/> [24.09.2021]

The most elaborate part of this editorial dashboard is the “items to moderate” section. It features a faceted search interface similar to the default one available to the normal users, enriched with additional facets that allow the Moderators to filter the items based on their status and other curation or meta information. Especially, the items can be filtered by the curation flags set by the automatic checks in the Python notebooks, representing the crucial part of the communication mechanism between automatic and manual curation processes. By combining the available filters, Moderators can easily narrow down a specific small subset of the items to concentrate on, e.g. all the Tools & Services from TAPoR, which feature a broken link.

4.5.4. Vocabulary Management

As described in section 3 *Data Model*, concept-based properties and controlled vocabularies are central elements of the Marketplace design and interoperability. Many of the properties that describe entities in MP use dedicated controlled vocabularies and the data model supports the notion of Vocabularies consisting of Concepts and provides the means to restrict the value ranges of individual dynamic properties to selected Vocabularies.

While some of the vocabularies can be considered “closed”, i.e. a predefined set of concepts, in most cases the system needs to allow for new concepts to be admitted to the vocabulary continuously, however in a controlled manner. New concepts can be introduced by Contributors/Moderators in need of a new term when editing an MP-item, but most often new terms are encountered during the automated ingest of data from existing sources. In general, human intervention is required to determine if the new term represents a new concept, is just another lexical representation of an existing concept, or should be disregarded altogether. This is at odds with the automatic ingestion process, prompting a need to disentangle the *items ingestions or curation* processes from the *vocabulary curation* process, allowing them to run asynchronously, while at the same time ensuring that newly created concepts can be used right away to describe MP-items.

This has been accomplished by introducing a status for the concepts, where newly encountered terms can be created as “candidate” concepts by the ingestion script, so that they can be used immediately, while keeping them in a kind of “quarantine” for the Moderators to identify them easily, and decide on their final admission to the vocabulary.

In the initial system architecture the Vocabulary Management was foreseen as a separate component, the main rationale being to avoid reimplementing functionality, given existing mature vocabulary management tools that could be readily reused, most notably VocBench⁵⁰. On the downside, this configuration implies the need for synchronisation of vocabularies between the vocabulary management component and the MP-core. During the first phase of the project, the vocabularies have

⁵⁰ VocBench vocabulary management tool: <http://vocbench.uniroma2.it/> [13.10.2021]

been managed by PoolParty Taxonomy Server⁵¹ and manually synchronized with the MP-backend. This setup turned out problematic, hampering the dynamic evolution of vocabularies, and the handling of newly encountered/proposed terms/concepts. Three possible alternative implementation options have been considered:

- A. “helper” twin properties for concept-based properties, storing candidate concepts until the next vocabulary synchronisation between the vocabulary manager and the MP-core;
- B. an indirection approach - there still is an external component for managing the vocabularies, however the synchronisation between this external component and the MP-backend is handled by the MP-backend, so that both ingestion scripts and curators (Moderators and Administrators) can in their basic routines restrict themselves to interaction with the marketplace. MP accepts new concepts and passes them on as candidate concepts to the external vocabulary management component, where they are handled accordingly in a separate vocabulary curation process.
- C. an integrated approach where whole vocabulary management is part of the MP-core.

Approaches B and C both require the following functionalities to be supported by the system:

- CRUD operations on Vocabularies
- Vocabulary curation as CRUD operations on Concepts supporting main features of the SKOS data model, i.e. semantic relations between concepts and a set of qualified labels per concept (preferred, alternative, hidden, language-wise)
- Bulk ingest of (concepts of) an existing vocabulary
- Associate Vocabularies with corresponding property types.

Even though indirection has been chosen as the preferred solution, it was delayed in the implementation. At the same time the practical handling of the vocabularies revealed the heterogeneous nature of vocabulary management, with vocabularies coming from different sources in different forms. This finding yielded a final setup, where there is no tight integration between MP-core and an external vocabulary management component. Rather the main/default operations are supported by the API of the MP-core component, there is minimal functionality and the version of vocabularies in the system is considered the source of truth. For the case when major curation effort is needed for a given vocabulary, it is exported in SKOS format, imported into a separate vocabulary management tool, curation is performed and the new version of the vocabulary is again exported in SKOS and reimported into MP via the API, making SKOS the primary interoperability mechanism.

⁵¹PoolParty Taxonomy server used for the Marketplace vocabularies: <https://sshoc.poolparty.biz/Vocabularies.html> [08.10.2021]

This setup required, following adjustments to the system, is as follows:

- adjustment of the data model, enriching the representation of Concepts covering the main aspects of the SKOS data model
- enhancing the API of MP-backend correspondingly to the richer data model, and introducing support for candidate concepts
- adaptations of the ingest pipelines to rely on MP as the authoritative source of vocabularies, and to push candidate concepts.
- introducing edit forms to create new ("candidate") concepts.

4.6. Extraction module

The extraction task had been foreseen as a potentially very fruitful, but experimental, feature of the system in the project proposal stage and correspondingly was described in the Description of Work. While the idea to extract structured information from publications and other textual material is very compelling, potentially yielding a host of new information, the extraction task is an exceedingly challenging one, rendering the outcome quite uncertain.

Correspondingly, this line of action has been kept independent from the main development and data processing task areas. Initial experiments have been conducted already in a very early stage of the project with a simple dictionary-based approach. That is a list of known tools coming from one of the primary sources - TAPoR - has been matched against a set of articles or conference abstracts (DH conferences 2015-2020) and against the Programming Historian lessons⁵² to offer a first idea of the feasibility and actual usefulness of the proposed task. These first experiments, implemented in a simple Java-based application named ToolXtractor⁵³ have been described in a series of white papers⁵⁴.

Next to these lexicon-based experiments, some machine learning and training of Named Entity Recognition (NER)⁵⁵ model approaches have also been developed with the idea that the extraction part would be integrated to the ingest process. The extraction module consists of two major parts, the Named Entity Recognition model and the extraction pipeline, where the NER model is integrated and used to extract new tool and relation candidates.

⁵² Programming Historian: <https://programminghistorian.org/> [08.10.2021]

⁵³ ToolXtractor: <https://github.com/lehkost/ToolXtractor/> [08.10.2021]

⁵⁴ See Which DH Tools Are Actually Used in Research?: <https://weltliteratur.net/dh-tools-used-in-research/>; DH Tools Mentioned in "The Programming Historian": <https://weltliteratur.net/dh-tools-programming-historian/> & Tools Mentioned in DH2020 Abstracts: <https://weltliteratur.net/tools-mentioned-in-dh2020-abstracts/> [08.10.2021]

⁵⁵ Named Entity Recognition: https://en.wikipedia.org/wiki/Named-entity_recognition [11.11.2021]

4.6.1. Named Entity Recognition Model

This section describes how machine learning, based on named entity recognition models trained on publications containing tool names, can be used to extract candidates for new tools that are not yet ingested in the marketplace.

- **Pre-processing:** Two Python scripts have been written to extract sentences from the conference abstracts (DH conferences 2015-2020) as the context for the extraction, the first script converts the original files in TEI format⁵⁶ to Plain Text files with one sentence per line. The second script converts the Plain Text files to JSONL⁵⁷ format.
- **Dataset:** Annotation tool Prodigy⁵⁸ is used to collect annotated data samples. The dataset currently consists of 1538 sentences annotated by two groups of annotators.
- **Model Training and Evaluation:** The annotated sentences with tool names have been used to train and evaluate the NER model. By using Prodigy's train recipe and a model pre-trained on the whole corpus, the resulting trained NER model has an F-score⁵⁹ of 91.622%.
- **Managing training pipeline and Named Entity Recognition models:** It is crucial for the reproducibility and maintainability of the model to track changes to the following artifacts:
 - Dataset might change upon manual evaluation of false-positives or false-negative cases.
 - Pre-processing pipeline might undergo some changes to deal with different formats
 - NER models should be retrained if either dataset or pipeline gets changed.

All three aspects are tracked and version controlled by using DVC⁶⁰, an extension of the git version control system with improved support for large datasets and complex composed data structures - which are typical for machine-learning tasks (training data, configuration, trained model, etc.). A DVC Pipeline is defined by specifying the pre-processing and training steps and their inputs and outputs as dependencies. The outputs are then stored on Google Cloud⁶¹ storage.

4.6.2. Extraction Pipeline

The scope of extraction has been limited to publications which are already ingested in the SSHOC Marketplace and have a valid link to the PDF file of the publication in the 'accessible_at' property. The extraction pipeline is defined as a DVC pipeline and consists of 5 tasks.

- **Publication retrieval:** SSHOC Marketplace is queried for publications containing valid PDF file links in the 'accessible_at' property, using a custom written Python client for the SSHOC

⁵⁶ Text Encoding Initiative: <https://tei-c.org/> [11.11.2021]

⁵⁷ JSON Lines: <https://jsonlines.org/> [19.10.2021]

⁵⁸ Prodigy: <https://prodi.gy/> [14.10.2021]

⁵⁹ F-score: <https://en.wikipedia.org/wiki/F-score> [11.11.2021]

⁶⁰ DVC: <https://dvc.org> [14.10.2021]

⁶¹ Google Cloud: <https://cloud.google.com> [14.10.2021]

Marketplace Rest API. The output is a list of publication objects with links to the publication file, and is saved as a pickle file⁶². A total of 2990 publications existed in the MP at the point of initial tool extraction execution, and 478 publications had a valid PDF file link.

- **Publication download:** the output from the previous task is used to download the actual PDF publications. The downloaded files are stored in a separate output directory. The output of the task is a mapping of publications to filenames and is stored as a pickle file.
- **Convert to XML:** this task converts the downloaded PDF files to XML/TEI using the grobid tool⁶³.
- **Tool candidate extraction:** The already trained NER model described in the previous section is applied to each sentence of the XML/TEI files and returns a list of candidate tool names. These tools are validated for relevance by checking in the MP if the tool name exists. There are two output artifacts from this stage. The first one is the 'publication to tool candidates' mapping and the second one is the list of tool names that are suggested by the NER model, but do not exist in the SSHOC Marketplace. Both are stored as pickle files.
- **Relation Ingestion:** The output from the previous task is used to add the relation '*Publication - mentions -> Tool*' to the MP. All other candidates that are suggested by the model but are not present in the marketplace are passed on to the Curation team for further investigations, and might be semi-automatically or manually ingested as new tool names.

The extraction pipeline can be either executed manually or periodically by using a Gitlab CI/CD pipeline.

4.7. Deployment and configuration of the system

The system consists of multiple components deployed as separate containers in a dockerized environment:

- backend component consisting of:
 - core application
 - Solr index
 - PostgreSQL database
- frontend component

The components can be easily deployed using the industry standard for light-weight virtualisation Docker⁶⁴.

The system is currently deployed in three instances on the servers run by DARIAH/OEAW:

- development - used by developers for testing new functionality
- staging - instance based on the latest version of the code and data model.

The ingests have been redirected to run against this instance, thus this instance also contains

⁶² More about pickle and pickle files: <https://docs.python.org/3/library/pickle.html> [11.11.2021]

⁶³ Grobid tool: <https://github.com/kermitt2/grobid> [14.10.2021]

⁶⁴ See Docker website: <https://www.docker.com/> [13.10.2021]

the latest data. It will replace the production instance for the final release, once all the ingestion and curation workflows have been tested and proven stable.

- production - publicly available stable instance based on the beta release

During the project the systems have been deployed using the container management tool Portainer⁶⁵, though the deployment will be moved to a Kubernetes⁶⁶ cluster by the end of the project. Initially this will be on a minimal cluster maintained by DARIAH/OEAW. As Kubernetes is the most common state-of-the-art container-orchestration solution used in cloud computing, a Kubernetes-compatible setup will stream-line the deployment and hosting of the system to external cloud providers, most notably e-Infrastructures offering their services within the EOSC. This is inline with the strategy to have the deployment and hosting under the project team's control during the period of heavy development, but have the system prepared to be readily hostable by professional cloud computing providers.

A continuous integration and deployment setup is in place, where pushing changes to corresponding branches in the git repository trigger a set of tests to be run and upon successful completion a new version of the application is automatically deployed.

There are a number of dynamic parts of the data model, which require configuration or initial population upon installation of a new naked/empty instance. These "system data" are maintained as YAML-configuration files as part of the backend source code and are applied during initialisation of the system. This includes:

- Dynamic properties or property-types, e.g. activity-type, keyword
- Actor roles, e.g. contributor, author, funder
- Item relation types, e.g. is-documented-by, mentions
- Concept relation types, e.g. related, broader, sameAs
- Sources, e.g. TAPoR, Programming Historian
- Actor sources, e.g. ORCID, DBLP, Wikidata

Vocabularies, though also part of the system data, are not loaded automatically on application startup anymore and need to be ingested separately. Similarly, if changes to the system data are needed for an existing system with a populated database, corresponding API calls can be applied by the Administrator.

To ensure data consistency and prevent data loss due to evolution of the data model, the Liquibase⁶⁷ tool is used to manage the database schema migrations. Corresponding configuration files for the schema migration are also maintained as YAML files (as part of the source code of the backend component) and are executed only once during application startup, before loading of the system data.

⁶⁵ See Portainer tool: <https://www.portainer.io/> [13.10.2021]

⁶⁶ Kubernetes: <https://kubernetes.io/> [11.11.2021]

⁶⁷ See Liquibase website: <https://www.liquibase.org/> [13.10.2021]

4.8. Application Programming Interface

As described in section 4 *Implementation*, the REST API exposed by the server application is the only way the client or any other component can interact with the data, decoupling the backend and frontend and making explicit the contract between server and client.

The API is described using Swagger⁶⁸ OpenAPI documentation.⁶⁹ The endpoint of the SSH Open Marketplace API is: <https://marketplace-api.sshopencloud.eu/api/>.

The API responses are in JSON and follow the Marketplace data model. For example, one can request a description of all the *tools and services* that the SSH Open Marketplace provides at:

<https://marketplace-api.sshopencloud.eu/api/tools-services>

and following pages:

<https://marketplace-api.sshopencloud.eu/api/tools-services?page=2>

The API is open for anyone to use. All read methods (HTTP GET method) are accessible anonymously, whereas for write operations (POST/PUT/DELETE) authentication is required, using the Bearer or token authentication method⁷⁰, in which first a JSON Web Token (JWT) has to be obtained at the authentication endpoint, and then sent as an Authorization header when connecting to other endpoints⁷¹.

5. EOSC Integration

Given the general goal of the whole SSHOC project developing the SSH part of the EOSC, it is crucial to ensure that the outcomes of the project will become part of this new evolving ecosystem, which for the SSH Open Marketplace as a service means primarily technical integration with the EOSC core services, the most prominent ones being the EOSC Portal Catalogue and Marketplace, Authentication and

⁶⁸ See Swagger tools: <https://swagger.io/> [12.10.2021]

⁶⁹ API Swagger documentation for the public instance of the Marketplace:
<https://marketplace-api.sshopencloud.eu/swagger-ui/index.html?url=/v3/api-docs> [12.10.2021]

⁷⁰ See the following resources for more details on the method:
<https://swagger.io/docs/specification/authentication/bearer-authentication/> and
https://en.wikipedia.org/wiki/JSON_Web_Token [14.10.2021]

⁷¹ See Bearer authentication (also called token authentication) specification:
<https://swagger.io/docs/specification/authentication/bearer-authentication/> [12.10.2021]

Authorization Infrastructure (AAI), Accounting, Monitoring and Helpdesk⁷². The SSH Open Marketplace has already successfully implemented integration with the EOSC AAI and is in the process of onboarding to the EOSC Portal. Integration with other EOSC core services is being considered.

However the SSH Open Marketplace, beyond being another service to become part of the EOSC ecosystem, is also a domain-specific discovery platform and a thematic catalogue. Finding the right resources in the vast, complex ecosystem is recognized as a major challenge, and there are numerous efforts to map various sub-areas of the scientific resources landscape. At the top, the EOSC Portal Catalogue and Marketplace and the OpenAire catalogue are emerging as the primary entry points for discovering services, resources and research outcomes. The position of the SSH Open Marketplace in this complex network of discovery solutions is detailed in section 5.3.

5.1. EOSC AAI

The federated identity management or AAI, is maybe the most critical core service of EOSC, given its role in establishing trust between systems and users and radically simplifying for the researchers the personalized access to the growing set of resources. Therefore the SSH Open Marketplace has been integrated with the EOSC AAI. This allows both for simple and reliable user management on the side of the Marketplace, and minimizing the entry barrier for users interested in getting actively involved, community involvement being one of the central propositions of the Marketplace.

Because the EOSC AAI was not available to the SSH Open Marketplace team at the beginning of the project, the EGI Check-In service has been used for the development environment. With the recent start of the EOSC Future project - in April 2021 - the management of the EOSC AAI changed and all communication and updates of configuration have to be handled via the EOSC Portal Helpdesk⁷³. The three SSH Open Marketplace environments are now using EOSC AAI (Production).

Technically, the authorization relies on the OAuth2 protocol. From the login page the user is redirected first to the WAYF (Where are you from) interface, where they can pick their home organization. Then the user is redirected again to the login page of the home organization and upon successful authentication is redirected back to the originating application (i.e. the Marketplace in this case) together with an implicit grant token as an URL fragment. This token is valid for only 30 seconds, during which time the frontend has to call the method PUT /api/oauth/token with this grant token to retrieve the actual JWT token in the Authorization header, which has to be then included in every subsequent request. This whole procedure is governed solely by redirects in the client's browser with no direct communication taking place between the server-side of the application and the identity provider.

⁷² For more details on the EOSC Architecture and the EOSC Core functions and services, see the EOSC Strategic Research and Innovation Agenda (SRIA) - <https://www.eosc.eu/sria> [14.10.2021] or the EOSC architecture working group view on the minimum viable EOSC - <https://op.europa.eu/s/s26Y> [14.10.2021].

⁷³ EOSC Portal Helpdesk: <https://eosc-portal.eu/helpdesk> [12.10.2021]

In the case that the user has logged in for the first time, they will be redirected to a registration form with prefilled “display name” and “email” as obtained from the identity provider to be but confirmed and with further information to be completed manually, as well as to accept the privacy policy. Thus while the authentication is done externally, there is still a local representation of the user in the system with additional information.

5.2. Onboarding SSH Open Marketplace into the EOSC portal

At the moment, the primary mode of integration of specialized services with EOSC is the “onboarding” of these services to the EOSC portal; essentially making them “registered” and listed in the EOSC catalogue, thus known and visible “citizens” of the EOSC ecosystem.

Onboarding services into the EOSC portal assists in implementing the SSHOC long-term strategy to ensure availability and findability of its services. As such, there is a concerted effort moderated by the project coordinator to identify mature services in their individual work packages and to ensure that these services are onboarded, including the SSH Open Marketplace. The onboarding should take place/be performed before the end of the project, however only after production release of the service, this being a precondition to onboarding.

EOSC portal distinguishes between two main entities, Providers and Resources. These are formalized in an interoperability framework called EOSC Profiles which defines the common data models and specifications for EOSC entities (such as the Providers Profile⁷⁴ and the Resource Profile⁷⁵) to be interoperable with its systems and others in the Ecosystem. These profiles allow a systematic automated management of the EOSC resource information from the point of metadata being entered by the service or resource provider during EOSC onboarding.

The rather complex onboarding process has been designed in a series of projects (such as eInfraCentral, EOSC hub, EOSCPilot, OpenAIRE-Advance, CatRIS and EOSC-Enhance) as well as the results produced by the EOSC Working Groups (e.g. Rules of Participation⁷⁶). This rich experience resulted in the EOSC onboarding process being well tested and rolled out successfully. The onboarding process essentially boils down to two stages: onboarding, or registering first the Provider entity and then subsequently registering the Provider’s “Resource”. While at the moment the resource profile is

⁷⁴ EOSC Providers Profile: <https://eosc-portal.eu/providers-documentation/eosc-provider-portal-provider-profile> [08.10.2021]

⁷⁵ EOSC Resource Profile: <https://eosc-portal.eu/providers-documentation/eosc-provider-portal-resource-profile> [08.10.2021]

⁷⁶ See outputs of the EOSC Rules of Participation Working Group - <https://www.eoscsecretariat.eu/working-groups/rules-participation-working-group/eosc-rop-outputs> [14.10.2021] - and the newly established Rules of Participation Compliance Monitoring Task Force - <https://www.eosc.eu/implementation-eosc> [14.10.2021] - for more details.

more geared towards services, there is ongoing work happening in the EOSC-Future project to extend the profile to cover other resources, like data sources and research data. Another improvement within EOSC-Enhance was the implementation of semi-automatic onboarding using the Providers API⁷⁷. Even though EOSC-Enhance is nearing completion in November 2021, further evolution/development of the onboarding processes and Portal development is to be expected, as the EOSC-Enhance project is handing over the work on the EOSC Portal to the EOSC Future project.

This onboarding is guided by the corresponding profiles, which are meanwhile available as registration forms in the EOSC Provider Portal. These are rather elaborate operations and contain a number of questions that may need extra consultation. Therefore, prior to commencing the actual registration procedure, it is recommended that a copy of the profiles be shared and consulted with stakeholders of the services, to allow for due preparation of the necessary information. This includes especially the clarification of the question “who is the *Resource Organisation* and/or *Provider*”, and the related question of commitment to provisioning and managing of the service on the portal. This topic has been discussed extensively both in the WP7 group, as well as in the broader scope of the SSH Science Cluster and the question of the long-term sustainability of the outcomes of the SSHOC project is taking shape at the time of writing⁷⁸. A Memorandum of Understanding between some of the main partners of the project will ensure the continuity of the collaboration after the end of the project, and dedicated services agreement(s) to clarify responsibilities among the SSH Open Marketplace *Organisations* and *Providers* as well as the financial support to be provided are under negotiation for some of the SSHOC outputs. Three ERICs involved in SSHOC are interested in sustaining the SSH Open Marketplace and some service providers have also notified their interest in maintaining the service after the end of the project.

5.3. Research community portals in the EOSC

Part of the work conducted during the SSHOC project was to monitor EOSC developments to understand what would be the role of the SSH Open Marketplace, as a discovery portal answering SSH communities needs, in this huge and complex growing ecosystem.

Grounded in community needs and requirements, the SSH Open Marketplace has been developed to be as user-friendly as possible for any users. In line with the development that there has to be many entry points into the vast and complex EOSC ecosystem, custom tailored to the specific needs of individual research communities, the SSH Open Marketplace has the potential to represent the main entrance for the SSH Science Cluster. In that sense, it is also foreseen that the SSH Open Marketplace will become part of the EOSC-Exchange, understood as the set of federated resources registered to the

⁷⁷ <https://providers.eosc-portal.eu/openapi>

⁷⁸ See for example SSHOC D7.5 Marketplace Governance published early October 2021.

EOSC by Research Infrastructures and Science Clusters to serve the needs of research communities⁷⁹. Next steps toward integrated discovery in the EOSC context (to which the SSH Open Marketplace could contribute) have to be aligned with the research data cataloguing efforts done in the five Science Cluster projects⁸⁰, as well as the outcomes of the EOSC-Enhance project⁸¹. Furthermore, as the options for integration of thematic catalogues in the EOSC catalogue are becoming clearer⁸² concrete avenues to ensure interoperability and integration of the SSH Open Marketplace within this ecosystem should rapidly emerge and surely represent the main future line of work to further develop the SSH Open Marketplace and ensure its sustainability as one of the SSH Cluster services in the EOSC.

6. Conclusion

This deliverable describes the successful implementation of the SSH Open Marketplace following the system specification (D7.1). In the iterative implementation process of the last two years, with major milestones being the minimum viable product, alpha and beta releases of the application, numerous adjustments and refinements to the original plan were introduced, but both the overall system architecture as well as the central principles of the data model proved right.

A tight collaboration between different dimensions of the work, such as the user requirements, the technical implementation, the interoperability and curation aspects - reflected in the WP tasks distribution - has been crucial for the success of the SSH Open Marketplace creation.

The SSH Open Marketplace is implemented, in line with the EOSC developments, as a resource catalogue, or discovery platform, to serve the Social Sciences and Humanities. Seeing the developments at the EOSC level, where the notion of “resource” is being gradually expanded beyond services to encompass datasets, training materials and other information objects, is gratifying and a confirmation of the SSH Open Marketplace path and data model.

Despite developing the SSH Open Marketplace against a very dynamic environment - the evolving EOSC ecosystem - the successful implementation of this SSH discovery portal is one of the concrete and tangible results of the SSHOC project and a good sign for the ongoing integration of the social scientists

⁷⁹ Next to the EOSC-Core and to the EOSC-Federation, the EOSC-Exchange is one of the main components of the EOSC Architecture.

⁸⁰ See Appleton, Owen, Petzold Andreas, Graf, Kay, Goble, Carole, Fischer, Frank, Richter, Tobias, & Willems, Marieke. (2020, November 16). Thematic Discovery Marketplaces for the European Open Science Cloud. Realising the European Open Science Cloud. Towards a FAIR research data landscape for the SSH and beyond., Online. Zenodo. <https://doi.org/10.5281/zenodo.4277601> [13.10.2021]

⁸¹ See Carole Goble, & Nick Juty. (2021). Analysis of existing research data cataloguing efforts towards integrated discovery. <https://doi.org/10.5281/zenodo.4693217> [13.10.2021]

⁸² See Workshop Incorporating National and Thematic Service and Resource Catalogues into the EOSC (esp. “Agreements & Policies for Catalogues” Owen Appleton (EOSC Future)) <https://eosc-portal.eu/events/incorporating-national-and-thematic-service-and-resource-catalogues-eosc> [13.10.2021]

and humanists needs' and resources into the EOSC.

7. References

- Appleton, Owen, Petzold Andreas, Graf, Kay, Goble, Carole, Fischer, Frank, Richter, Tobias, & Willems, Marieke. (2020, November 16). Thematic Discovery Marketplaces for the European Open Science Cloud. Realising the European Open Science Cloud. Towards a FAIR research data landscape for the SSH and beyond., Online. Zenodo. <https://doi.org/10.5281/zenodo.4277601>
- Barbot, Laure, Moranville, Yoan, Fischer, Frank, Petitfils, Clara, Ďurčo, Matej, Illmayer, Klaus, Parkoła, Tomasz, Wieder, Philipp, & Karampatakis, Sotiris. (2019). SSHOC D7.1 System Specification - SSH Open Marketplace (1.0). Zenodo. <https://doi.org/10.5281/zenodo.3547649>
- Barbot, Laure, Moranville, Yoann, Buddenbohm, Stefan, Illmayer, Klaus, & Ďurčo, Matej. (2020). MS42 Marketplace – alpha release (v1.0). Zenodo. <https://doi.org/10.5281/zenodo.4585700>
- Barbot, Laure, Fischer, Frank, Illmayer, Klaus, Ďurčo, Matej, König, Alexander, Van Uytvanck, Dieter, & Larrousse, Nicolas. (2020). MS.43 - Marketplace - beta release (1.0). Zenodo. <https://doi.org/10.5281/zenodo.4785194>
- EURISE Technical Reference: <https://technical-reference.readthedocs.io>
- Fischer, Frank, Moranville, Yoann, Barbot, Laure, Capelli, Laurent, Ngo, Virginie, Dumouchel, Suzanne, Blotière, Emilie, Ďurčo, Matej, Van Uytvanck, Dieter, König, Alexander, Gingold, Arnaud, Nowak, Aleksandra, & Parkoła, Tomasz. (2020, November 25). Exploring the SSH data landscape: thematic discovery portals in the EOSC. REALISING THE EUROPEAN OPEN SCIENCE CLOUD Towards a FAIR research data landscape for the social sciences, humanities and beyond. Zenodo. <https://doi.org/10.5281/zenodo.4290599>
- Goble, Carole & Juty, Nick. (2021). Analysis of existing research data cataloguing efforts towards integrated discovery. <https://doi.org/10.5281/zenodo.4693217>

List of Figures

- [Fig. 1 - Data Model v 1.5](#)
- [Fig. 2 - Statuses for items](#)
- [Fig. 3 - System Architecture Diagram as introduced in D7.1 System Specification](#)
- [Fig. 4 - Update of the system architecture](#)
- [Fig. 5 - low-fidelity sketches of the search result page](#)
- [Fig. 6 - Screenshot of the Gephi tool - Item detail view](#)
- [Fig. 7 - Overview on the different components of the ingestion workflow](#)
- [Fig. 8 - Ingestion workflow](#)
- [Fig. 9 - Screenshot of the Data Acquisition DPUs in Pool Party](#)
- [Fig. 10 - JSON to JSON transformation on the JOLT demo site for DARIAH-Campus source](#)
- [Fig. 11 - Screenshot of the edit form - "3DHOP - How To" item](#)
- [Fig. 12 - screenshot of the "My account" homepage for Administrator](#)

List of Tables

- [Table 1 - Vocabularies used in concept-based properties](#)
- [Table 2 - Components of the system architecture: foreseen vs. implemented](#)

8. *Annex 1 - EURISE Software Quality Checklist applied to the SSH Open Marketplace*

General

- X Does the software have a descriptive name?
- X Is there a short high-level description of the software?
- X Is the purpose of the software clear?
- X Does the software exactly match its requirements?
- X Is the targeted audience of the software clear?
- X Has the software been tested by members of the target audience in respect of its usability?
- X Does the software (and its dependencies) use OSI approved licenses?
- X Is the software under version control?
- X Is there a website for the software?
- X Is the software's website mobile friendly to a certain degree? I.e. can it be accessed on a smartphone or tablet without hiding the most important information and features?
- X Are the user interface design and the software's website mindful of accessibility? I.e. does it consider e.g. a high contrast between colors for colorblind users, are there alternative texts for images that can be read by a screenreader, are texts easily resizeable, ...?
- X Does the software have a release mechanism?
- X Is the software available in packaged format or only sources?
- X Are maintainer and development status clear, including up to date and accessible contact information?
- X Are the requirements listed and up to date?
- X Is copyright and authorship clear and accessible?
- X Is there a contribution guide?

Documentation

- X Is there an accessible low-level guide for getting started?
- X Is there an accessible user guide?
- X Is there a full user documentation?
- X Does the user interface link to held references?
- X Are there examples, FAQs and tutorials?
- X Is there information stated about who to ask when a problem is not covered by the FAQ?
- X Are known issues documented and easily accessible for all user groups?
- X Can bugs/issues be reported easily by other developers and users?

Development

- X Is the development setup documented?
- X Is the build mechanism documented?
- X Does the build mechanism use a common single-command system (i.e. Maven)?
- X Is the software API documented?
- [*in progress*] Are all appropriate config options externalised and documented?
- X Does the code allow internationalisation (i18n)?
- [*no*] Is the software localised (l10n)? English is mandatory.
- X Is there a test suite?
- [*94% classes, 79% methods and 76% lines*] Is test coverage above 80%?
- X [*run locally by developers before commit*] Are the tests run on a regular and frequent basis, e.g. on commit/every night/...?
- X Do you have and stick to a policy for security by design?
- X Is the software portable?
- X Has the portability been tested?

Interoperability

- X Are file formats standard compliant and documented?
- X Is the API standard compliant?
- X Does it provide a monitoring endpoint?
- X Does it adhere to an interface style guide?
- X Does it use existing authentication systems (OAuth2/eduGain)?

Administration

- X Are software requirements such as operating system, required libraries and dependencies specified including versions?
- [*planned*] Are hardware requirements for CPU, RAM, HDD, Network specified?
- X Are there deployment instructions?
- [*in progress*] Is there a comprehensive and fully documented example configuration?
- X Is a startup script provided?
- X Are there troubleshooting guides?