



Building the Legal Knowledge Graph for Smart Compliance Services in Multilingual Europe

D4.4 Initial Implementation and Report of Data and Content Curation Services

PROJECT ACRONYM	Lynx
PROJECT TITLE	Building the Legal Knowledge Graph for Smart Compliance Services in Multilingual Europe
GRANT AGREEMENT	H2020-780602
FUNDING SCHEME	ICT-14-2017 - Innovation Action (IA)
STARTING DATE (DURATION)	01/12/2017 (36 months)
PROJECT WEBSITE	http://lynx-project.eu
COORDINATOR	Elena Montiel-Ponsoda (UPM)
RESPONSIBLE AUTHORS	Julián Moreno Schneider (DFKI), Georg Rehm (DFKI)
CONTRIBUTORS	Julián Moreno Schneider (DFKI), Georg Rehm (DFKI), Filippo Maganza (ALP), Sotiris Karampatakis (SWC), Víctor Rodríguez Doncel (UPM)
REVIEWERS	Andis Lagzdīņš, Ēriks Ajausks (TILDE), Víctor Rodríguez Doncel (UPM), Socorro Bernardos (UPM)
VERSION STATUS	V1.0 Final
NATURE	Other
DISSEMINATION LEVEL	Public
DOCUMENT DOI	10.5281/zenodo.3557757
DATE	29/11/2019 (M24)



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780602

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
0.1	First TOC Version	22.10.2019	Julián Moreno Schneider (DFKI), Georg Rehm (DFKI)
0.2	Second TOC based on reviewer's comments	03.11.2019	Julián Moreno Schneider (DFKI), Georg Rehm (DFKI)
0.3	Inclusion of content in Section 2	04.11.2019	Sotiris Karampatakis (SWC)
0.4	Inclusion of LKG content in section 2	05.11.2019	Víctor Rodríguez Doncel (UPM)
0.6	Including content con Section 3	06.11.2019	Filippo Maganza (ALP)
0.7	Including content in and finishing Section 3	07.11.2019	Julián Moreno Schneider (DFKI), Georg Rehm (DFKI)
0.9	Finishing Executive Summary, Introduction and Conclusions	08.11.2019	Julián Moreno Schneider (DFKI), Georg Rehm (DFKI)
1.0	Including final reviews and feedback	28.11.2019	Julián Moreno Schneider (DFKI), Georg Rehm (DFKI)

ACRONYMS LIST

BB	Building Block
BPMN	Business Process Model and Notation
CWM	Curation Workflow Manager
DAG	Directed Acyclic Graph
DCM	Document Manager
DS	Dataset
EntEx	Entity Extraction
GEO	Geolocation information extraction
GUI	Graphical User Interface
LKG	Legal Knowledge Graph
NER	Named Entity Recognition
RDF	Resource Description Framework
RelEx	Relation Extraction
RFP	Request for Proposal
SC	Scenario
SeSim	Semantic Similarity
StrEx	Document Structure Analysis
Summ	Summarization
TERM	Terminology extraction
TIMEX	Temporal Expression Analysis
ToClass	Topic Classification (Detection)
TRANS	Translation
UC	Use Case
URI	Uniform Resource Identifier
WM	Workflow Manager
WME	Workflow Manager Engine
WP	Work Package
WSD	Word Sense Disambiguation

EXECUTIVE SUMMARY

This report describes the initial implementation of the curation workflow manager (WM), which controls the workflows associated to every business use case (as defined in D4.1 [LynxD41]). This implementation is based on the definition provided in D4.2 [LynxD42] and D4.3 [LynxD43], in which we outlined four workflows: a common workflow, *LKG population*, and three use case specific workflows, namely *Contract Analysis (OLS)*, *Geothermal Project Analysis(DNVGL)*, and *Labour Law Question Answering (CuatreCasas)*.

The implementation of the curation workflow manager is based on the Camunda BPMN engine (<https://camunda.com/products/bpmn-engine/>), which allows the definition, implementation and execution of workflows inside the same tool. The main components of the curation workflow manager are:

1. **Workflow Manager Engine:** it is responsible for converting workflows in tasks for the workers.
2. **Workers:** they are responsible for the execution of tasks inside a workflow.
3. **Shared memory service:** it is the service that both the Workflow Manager Engine and the Workers use to share large data objects.
4. **CAMUNDA API:** it is a complete set of RESTful APIs useful to manage BPMN process definition [OMG2011], process instances and their history.
5. **PILOTS API:** it is a component of the WM which is responsible for the access to manage and execute workflows.
6. **Camunda Modeler:** it is a graphical user interface for defining workflows.

At the moment of writing this deliverable we have already implemented three workflows of the four defined in D4.3 [LynxD43].

- Legal Knowledge Graph Population
- Contract Analysis
- Geothermal Project Analysis

This report also describes the implementation of the document manager(DCM), which is responsible for the storage of the Legal Knowledge Graph and the documents once they have been processed through the different workflows. The document manager is implemented using Trellis Linked Data Platform (LDP) (<https://www.trellisldp.org/>), so that it extends the idea of the Knowledge Graph (KG) including also the documents. By utilizing the flexibility of JSON-LD and the capabilities of an LDP, the DCM is the main building component of the Lynx Legal Knowledge Graph (LKG) and where the LKG lies.

In the time left to finish the project, the following implementations will be completed:

- The workflow defined in D4.3 that is not yet implemented (*Labour Law* workflow).
- Some modifications in the curation workflow manager to support more efficient communication capabilities between the services.

TABLE OF CONTENTS

1	INTRODUCTION	5
1.1	PURPOSE OF THIS DOCUMENT	5
1.2	STRUCTURE OF THIS DOCUMENT	5
2	DOCUMENT MANAGER.....	6
2.1	ANNOTATION CONTROLLER.....	6
2.1.1	GET	7
2.1.2	POST	7
2.1.3	PUT.....	7
2.1.4	DELETE	8
2.2	LEGAL KNOWLEDGE GRAPH IN THE DOCUMENT MANAGER.....	8
3	CURATION WORKFLOW MANAGER	9
3.1	WORKFLOW MANAGER ENGINE.....	9
3.2	WORKERS	10
3.3	SHARED MEMORY SERVICE.....	11
3.4	CAMUNDA API.....	11
3.5	PILOTS APIS	11
3.5.1	POST – LKGP, GPA and CA	12
3.5.2	GET	13
3.6	GRAPHICAL USER INTERFACE FOR DEFINING WORKFLOWS	13
3.7	EXECUTION OF A WORKFLOW	13
4	CONCLUSIONS.....	15
	ANNEX 1 – IMPLEMENTATION DETAILS (CODE).....	16

TABLE OF FIGURES

Figure 1. Discrete methods available in the Annotation Controller.....7

Figure 2. Visualization of the LKG information ingestion process.....8

Figure 3. Architecture of the Camunda-based curation workflow manager.9

Figure 4. Image of the Gitlab repository of the Workflow Manager engine.....10

Figure 5. Screenshot of the Gitlab repository of the implemented Workers' types.....11

Figure 6. Visualization of the Swagger Open API Specification of the APIs defined in the Workflow Manager for the pilots.12

Figure 7. Example of a Workflow defined.13

TABLE OF TABLES

Table 1. List of repositories related to DCM and WM.....16

1 INTRODUCTION

A curation workflow manager (CWM) is needed in order to orchestrate all the different services of the Lynx platform because it has been designed using a microservice architecture. This manager is responsible for the execution of all workflows defined in Lynx, therefore it must be able to contact all the services, named Building Blocks (BB) in Work Package 3 (WP3), and datasets in order to implement the required functionality. The datasets are stored by another module, namely document manager (DCM), that is responsible for the storage of the Legal Knowledge Graph (LKG) and the enriched documents that are processed through the workflows.

The document manager is implemented in a flexible manner using different data storages (Elastic Search and the Trellis Linked Data Platform (LDP)), so it extends the idea of the documents to the level of a Knowledge Graph (KG). By utilizing the flexibility of JSON-LD and the capabilities of an LDP, the DCM is the main building component of the Lynx Legal Knowledge Graph (LKG) and where the LKG lies. Further information on the Legal Knowledge Graph can be found in the devoted section of D2.4, Data Management Plan [LynxD24].

The main components of the CWM are the different components that are responsible for orchestrating the execution, namely *Workflow Manager Engine*, the internal and external storage of information, namely *Shared Storage*, and the different elements for contacting the services involved in the workflows, namely *Workers*.

At the moment of writing this deliverable we have already implemented three workflows of the four defined in D4.3 [LynxD43].

- Legal Knowledge Graph Population
- Contract Analysis
- Geothermal Energy

1.1 PURPOSE OF THIS DOCUMENT

This report describes the initial implementation of the curation workflow manager and the document manager as components of the Lynx platform. The document is based on D4.3 [LynxD43], which defines the final version of the workflows, and aligned with D1.1 [LynxD11] and D4.1 [LynxD41], which define the requirements for the Lynx platform collected from the use case pilots.

1.2 STRUCTURE OF THIS DOCUMENT

Section 2 Document Manager describes the document manager which handles the stored information inside the platform. Section 3 Curation Workflow Manager describes the curation workflow manager as a component of the Lynx platform. Section 4 Conclusions concludes this deliverable. Annex 1 presents the implementation details of the curation workflow manager.

2 DOCUMENT MANAGER

The document manager (also referred to as DCM) is a central part of the Lynx platform in terms of the general platform capabilities. Its main functionalities include the storage of documents and their annotations; with special emphasis on keeping the synchronization among them, providing read and write access, and update of documents and annotations.

The document manager can be queried in terms of annotations (e.g. “which documents mention entity X”), and in terms of documents (e.g. “what are the contents/annotations of document X”). All queries to the DCM are executed via a REST interface. The interface includes a set of Create, Read, Update, and Delete (CRUD) APIs to manage the following resources within the Lynx platform: collections, documents and annotations.

The DCM has been designed with the intention of being an abstract entity with different possible implementation in order to respond to the different needs of the pilots, in terms of volume of data, speed of access, etc. The DCM, implemented in Java, has as central contract an interface (DocumentManager), which so far has had three different implementations: one based on files, one based on ElasticSearch¹ and one based on the Trellis Linked Data Platform². Although the first has been left only for informative purposes, the implementations based on ElasticSearch and on Trellis are fully functional and perfectly interchangeable. The adopted design pattern to manage these implementation has been the *dependency injection*, implemented with the Spring `@autowired`, and enabling the automatic detection of relationships between beans.

By having a representation of the documents in JSON-LD (namely, RDF), the Lynx documents are not only isolated elements but nodes in a graph; and the use of semantics to formalize the meaning of the classes and properties qualifies this graph to be called Knowledge Graph. The DCM is the main building component of the Lynx Legal Knowledge Graph (LKG) and where the LKG resides. Basic architecture and description of the core functionalities are already described in D1.4[LynxD14]. We document here a more detailed description of the Annotation Controller, the decision to use JSON-LD as the core data interchange format within the Lynx Platform and a description of how the DCM builds the Legal Knowledge Graph.

2.1 ANNOTATION CONTROLLER

The Annotation Controller is the endpoint of the DCM, which allows the manipulation of and the interaction with annotations of a document. It consists of 7 discrete methods (see Figure 1):

- 3 HTTP GET methods to retrieve one, all or a list of some annotations of a document
- 1 HTTP POST method to create new annotations on a document
- 1 HTTP PUT method to update annotations
- 2 HTTP DELETE method to delete annotations.

¹ <https://www.elastic.co>

² <https://www.trellisldp.org>

annotation		Annotation Controller	▼
GET	/collections/{collectionId}/documents/{documentId}/annotations	Gets all the annotations of a document	
POST	/collections/{collectionId}/documents/{documentId}/annotations	Creates new annotation(s)	
PUT	/collections/{collectionId}/documents/{documentId}/annotations	Updates annotations of a document	
GET	/collections/{collectionId}/documents/{documentId}/annotations/**	Gets an annotation	
DELETE	/collections/{collectionId}/documents/{documentId}/annotations/**	Deletes an existing annotation	
GET	/collections/{collectionId}/documents/{documentId}/annotations/list	Lists all the annotations of a document	

Figure 1. Discrete methods available in the Annotation Controller.

All annotations within the Lynx context are described in RDF (Resource Description Framework) by using the NIF2.1 vocabulary. Additionally, annotation IDs within the DCM are equal to the annotation URIs (Uniform Resource Identifier), thus uniquely identifiable. A complete documentation and testing environment can be found online³. The rest of the section describes the methods in brief.

2.1.1 GET

The HTTP GET methods of the DCM allow clients to access the annotations of a document.

The client may request a list of all annotations of a document either in RDF or JSON format. Additionally, a client may request all annotations of a document, or a particular annotation of a document, by specifying the particular annotation ID. Annotations can be provided in various RDF formats and in JSON.

2.1.2 POST

In order to create new annotations, a client may use the POST method of the Annotation Controller. All annotation producing services of the Lynx Platform are providing annotations in NIF2.1 format, using the TURTLE serialization of RDF, thus the method accepts input in TURTLE format. Multiple POST requests on the same document will be accepted as separate resources. The actual RDF content of the body request is then consolidated with the existing annotations.

2.1.3 PUT

The PUT method allows to update annotations of a document. The method accepts inputs as RDF (in TURTLE format). Existing annotations of the document will be replaced by the new content. The LDP backend of the DCM allows a more advanced approach in updating the annotations by accepting SPARQL Update queries, giving better flexibility on the client on how to update resources, but this is currently not accepted by the DCM.

³ <http://docmanagerldp-88-staging.cloud.itandtel.at/swagger-ui.html#/annotation>

2.1.4 DELETE

The DELETE method allows to delete annotations of a document. A client may delete all the annotations of a document or a particular annotation, by specifying the annotation ID (the annotation URI).

2.2 LEGAL KNOWLEDGE GRAPH IN THE DOCUMENT MANAGER

As stated in the proposal document, “Lynx aims to create a knowledge graph of legal and regulatory data towards compliance, in which heterogeneous data sources from different jurisdictions, languages and orders are aggregated and interlinked by a collection of advanced services”.

The DCM has an integral role in the construction and curation of the Legal Knowledge Graph within the LYNX platform (see Figure 2). As the DCM is implemented using an LDP server such as Trellis, basic metadata about the document itself are stored as triples natively –the implementation based on Elastic Search storing equivalently the JSON-LD syntax of RDF. Additionally, document structure metadata and various types of metadata such as subject, jurisdiction, language etc. are also triplified through the DCM at storing time. Ontology NIF v2.1 is used to describe structure metadata and a mashup of metadata specific ontologies are used for other descriptive, structural or administrative metadata. Moreover, annotations of each document are also described using the NIF v2.1 ontology. An overview of the data model used to describe documents and their metadata within Lynx can be found in <http://lynx-project.eu/data2/data-models>. Triples from all documents including data and metadata can be queried using the SPARQL endpoint provided by the Trellis platform, thus providing access to the LKG and ability to evaluate complex queries –the equivalent for the ElasticSearch implementation being made by periodic data exports, queryable in the endpoint <http://sparql.lynx-project.eu/>. The extensive usage of vocabularies as values for metadata or annotations enhances the value of the LKG and increases the interoperability of the system.

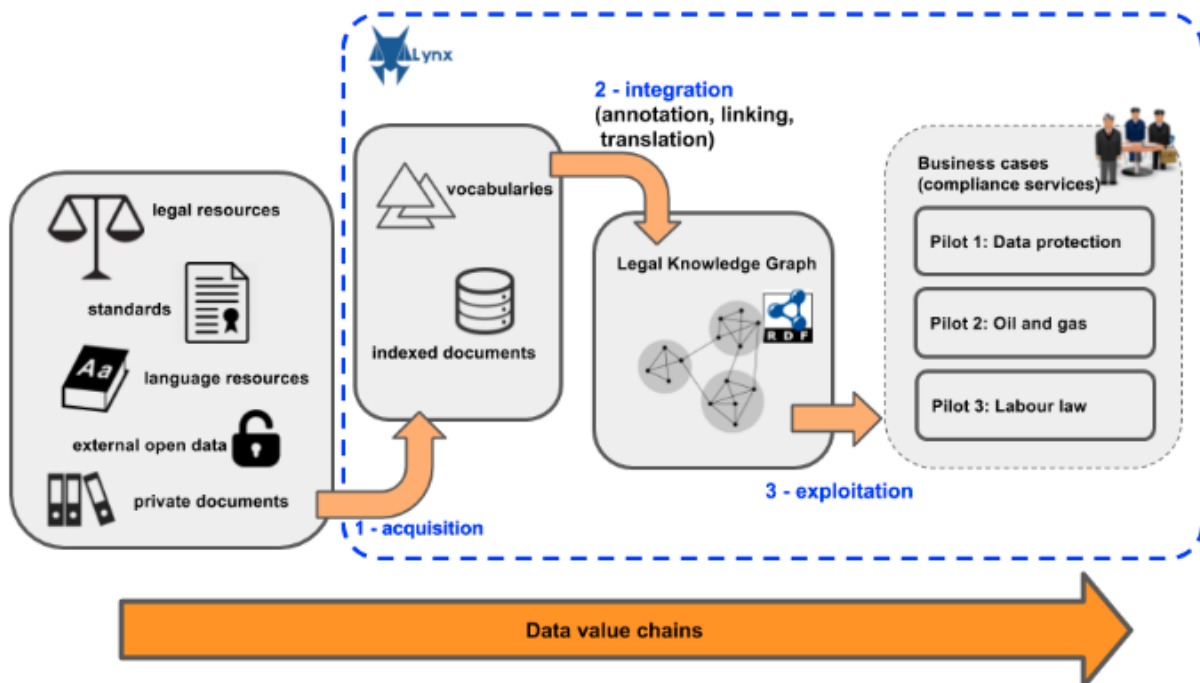


Figure 2. Visualization of the LKG information ingestion process.

3 CURATION WORKFLOW MANAGER

Using a microservice architecture enforces the usage of some kind of management in order to orchestrate the execution of the different services involved in more complex tasks. The combination of several functionalities from different services is defined as a workflow and the module responsible for orchestrating them is called curation workflow manager (CWM).

We have already published a paper about a generic workflow manager for curation technologies [Bourgonje2016], and two different papers describing the curation workflow manager of the Lynx platform [MorenoSchneider2018a, MorenoSchneider2018b].

At the beginning of WP4 we started the development of two different curation workflow managers –one based on RabbitMQ (<https://www.rabbitmq.com/>) and one based on Camunda BPMN engine (<https://camunda.com/>). Finally, we adopted the Camunda-based solution because it was in a more mature state, so only the Camunda-based version is described in this deliverable.

Figure 3 shows the current architecture of the CWM. Its main logical components are described in the following sections.

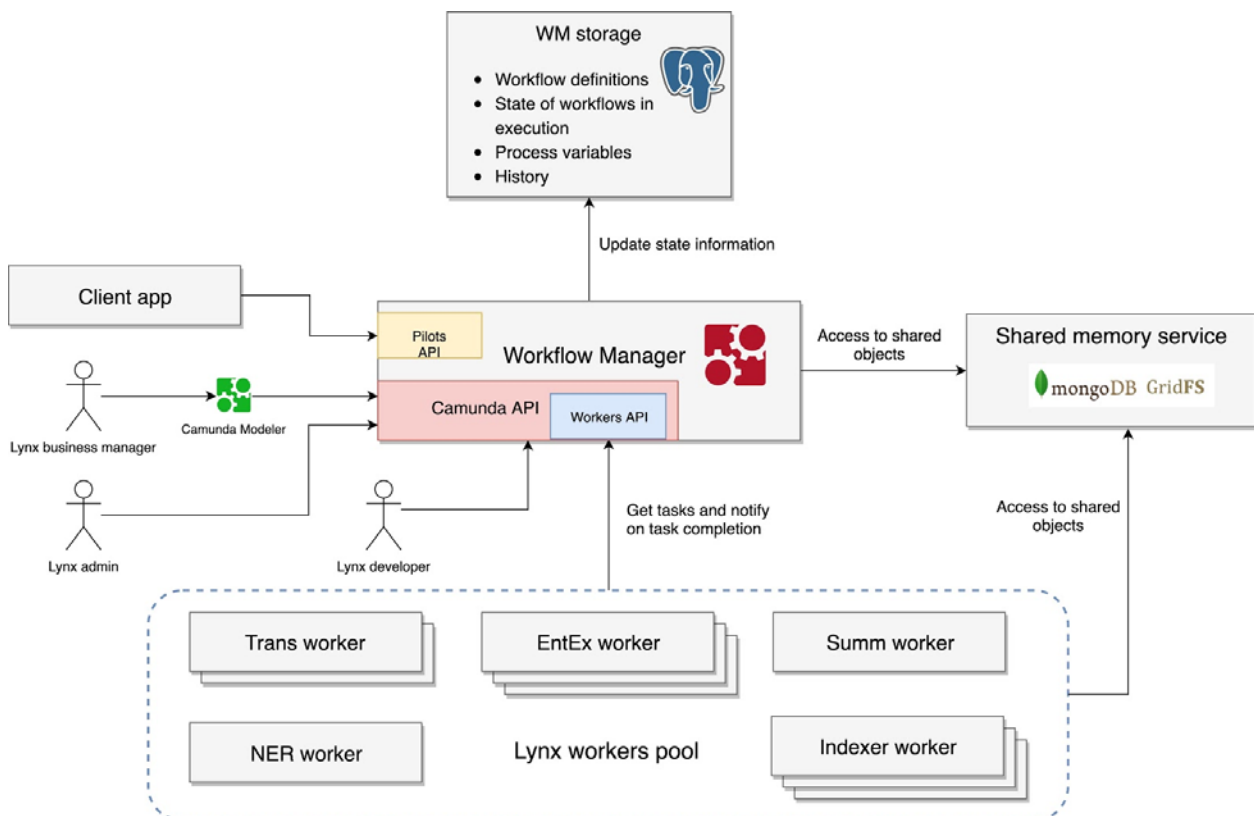


Figure 3. Architecture of the Camunda-based curation workflow manager.

3.1 WORKFLOW MANAGER ENGINE

The Workflow Manager Engine (WME) is responsible for converting workflows in tasks for the workers. Its implementation is based on the Camunda BPMN engine [BPMN2019]. The main concepts of this component are:

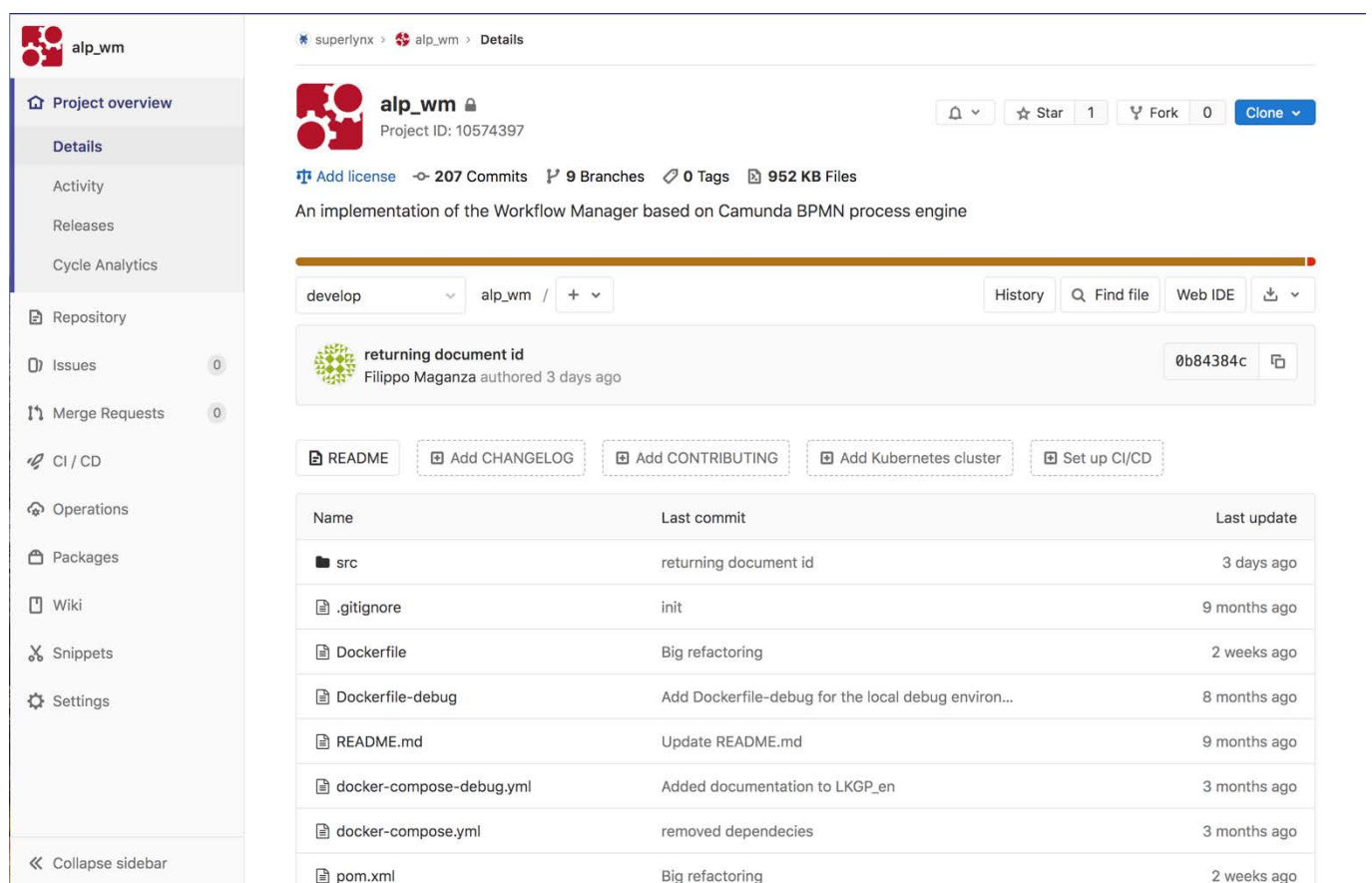
- **Workflow:** a direct acyclic graph whose nodes are associated with tasks;

- **Task:** an atomic unit of business logic; a task is associated with one and only one Lynx peripheral service;
- **Process:** a runtime instance of a workflow;
- **Job:** a runtime instance of a task.

It also provides a very complete REST interface for managing workflow executions and templates that is completely described in Section 3.5.

Apart from that, the WME uses some internal storage to save the different objects (workflows, tasks, processes and jobs) that are created during execution. All these elements are stored in the Workflow Manager Storage, which is implemented using a PostgreSQL database (<https://www.postgresql.org/>).

All the code of the WME can be found at https://gitlab.com/superlynx/alp_wm. This repository is private and accessible upon request, therefore an image of the repository is also provided in Figure 4.



The screenshot shows the GitLab repository page for 'alp_wm'. The left sidebar contains navigation options: Project overview, Details, Activity, Releases, Cycle Analytics, Repository, Issues (0), Merge Requests (0), CI / CD, Operations, Packages, Wiki, Snippets, and Settings. The main content area displays the repository details, including the project name 'alp_wm', Project ID '10574397', and statistics: 207 Commits, 9 Branches, 0 Tags, and 952 KB Files. A description states: 'An implementation of the Workflow Manager based on Camunda BPMN process engine'. Below this, there are options to 'Add license', '207 Commits', '9 Branches', '0 Tags', and '952 KB Files'. A commit history table is visible, showing the most recent commit by Filippo Maganza 3 days ago with the commit ID 0b84384c. Below the commit history, there are buttons for 'README', 'Add CHANGELOG', 'Add CONTRIBUTING', 'Add Kubernetes cluster', and 'Set up CI/CD'. A table lists the repository files and their last commit details.

Name	Last commit	Last update
src	returning document id	3 days ago
.gitignore	init	9 months ago
Dockerfile	Big refactoring	2 weeks ago
Dockerfile-debug	Add Dockerfile-debug for the local debug environ...	8 months ago
README.md	Update README.md	9 months ago
docker-compose-debug.yml	Added documentation to LKGP_en	3 months ago
docker-compose.yml	removed dependencies	3 months ago
pom.xml	Big refactoring	2 weeks ago

Figure 4. Image of the Gitlab repository of the Workflow Manager engine.

3.2 WORKERS

The Workers are responsible for the execution of tasks inside a workflow. Every task is identified by a name that refers to the related services like “*TimEx-LKGPpopulation*” or “*NER-ContractAnalysis*”. Workers can be easily scaled in and out using Kubernetes.

A worker is a piece of Java code that uses the Camunda External Task Client library (<https://docs.camunda.org/manual/7.9/user-guide/ext-client/>) to connect to the WME to obtain the tasks it has to execute. At the moment of writing this deliverable, there are four implemented types of workers. Each type of worker can be instantiated multiple times with different configuration files, i.e.,

each instantiated worker is responsible (based on its configuration) to connect to a different service, or even to the same service but with different parameters (for example, *lang=de* or *lang=en*). The four types of Workers (shown in Figure 5) have different functionalities:

- **document-translation-worker:** it is responsible to connect to the Tilde translation services.
- **document-enrichment-worker:** it is responsible to connect to one of the enrichment services inside the Lynx platform: NER, TIMEX, SUMM, WSID, EntEx, etc.
- **save-enriched-doc-in-LKG-worker:** it is responsible to save an enriched document inside the DCM.
- **create-enriched-document-worker:** it is responsible to create an enriched document.

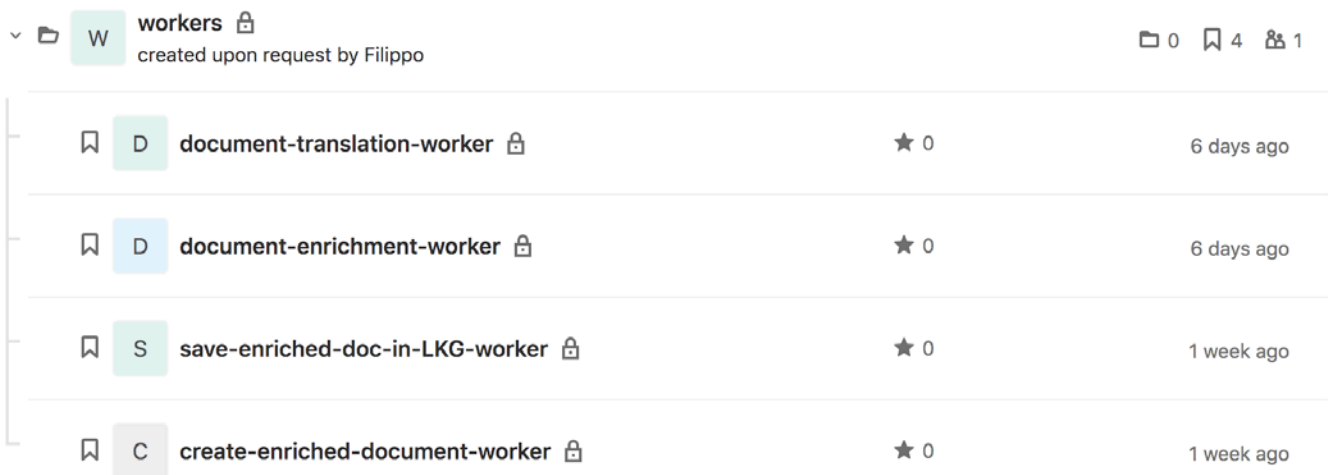


Figure 5. Screenshot of the Gitlab repository of the implemented Workers' types.

3.3 SHARED MEMORY SERVICE

The shared memory service is the service that both the Workflow Manager Engine and the Workers use to share large sized data objects. For instance, currently the WM uses it to share with the workers the documents that they have to process.

The shared memory service uses a MongoDB (<https://www.mongodb.com/>) database to store the information.

3.4 CAMUNDA API

The Camunda Rest Engine provides a complete set of RESTful APIs useful to manage Business Process Model and Notation (BPMN) process definitions, process instances and their history. We have decided to include some of these APIs in the Lynx Workflow Manager to extend its basic functionalities. As an example, if a maintenance of the Lynx platform is required, it will be possible to pause the workflows in execution for some time and then resume them when the maintenance is complete.

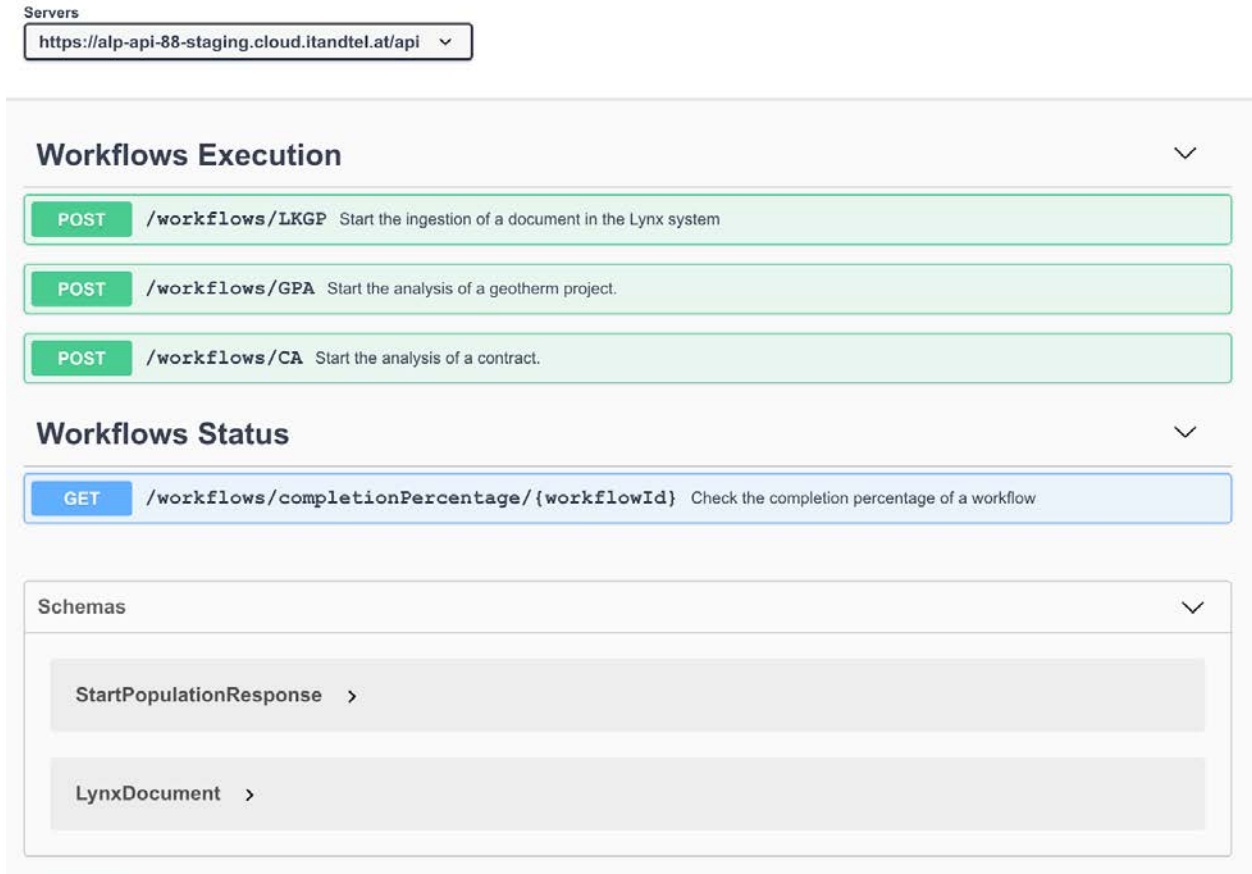
This interface will be available only to Lynx developers, administrators and business managers. A very detailed documentation is available at <https://docs.camunda.org/manual/latest/reference/rest/>.

3.5 PILOTS APIS

The Pilots API is a component of the CWM, which is responsible for the access to managing and executing workflows. It consists of 4 discrete methods:

- 3 HTTP POST methods to execute and manage pilot specific workflows (one method per pilot);
- 1 HTTP GET method to retrieve the current state of a concrete workflow.

These methods are shown in Figure 6 and described further in the document, but the complete information is online.⁴



The screenshot displays the Swagger Open API Specification for the Workflow Manager. At the top, there is a 'Servers' dropdown menu showing the URL 'https://alp-api-88-staging.cloud.itandtel.at/api'. Below this, the API is organized into three main sections:

- Workflows Execution:** This section contains three POST methods:
 - POST /workflows/LKGP:** Start the ingestion of a document in the Lynx system.
 - POST /workflows/GPA:** Start the analysis of a geotherm project.
 - POST /workflows/CA:** Start the analysis of a contract.
- Workflows Status:** This section contains one GET method:
 - GET /workflows/completionPercentage/{workflowId}:** Check the completion percentage of a workflow.
- Schemas:** This section lists two schemas:
 - StartPopulationResponse**
 - LynxDocument**

Figure 6. Visualization of the Swagger Open API Specification of the APIs defined in the Workflow Manager for the pilots.

3.5.1 POST – LKGP, GPA and CA

The HTTP POST methods of the CWM allow clients to manage workflow executions and templates. There is one method per use case:

- **LKGP** for the Legal Knowledge Graph Population workflow (see deliverable 4.3 [LynxD43]).
- **CA** for Scenario 1 Contract Analysis (see deliverable 4.3 [LynxD43]).
- **GPA** for Scenario 2 Geothermal Project Analysis (see deliverable 4.3 [LynxD43]).

The client may execute a concrete workflow with a specific request whose body must contain a valid LynxDocument JSON. In the parameters, it must specify the *collection* in which the document and its annotations are going to be stored. The response is a *StartPopulationResponse*, which corresponds to a confirmation that the workflows is running including a *workflodId*, which is needed to request its status using the GET method.

⁴ http://lynx-project.eu/doc/api/alp_wm.html

3.5.2 GET

The HTTP GET method of the Pilot APIs of the CWM allows clients to retrieve the current status of a concrete workflow execution. The client must provide the *workflowId* as part of the URL to call the method. The response will be a JSON one containing the relevant information about the current state of the workflow execution.

3.6 GRAPHICAL USER INTERFACE FOR DEFINING WORKFLOWS

The first general definition of workflow in the domain of the Lynx project was simply a direct acyclic graph where the nodes are tasks and the edges represent the order in which tasks must be executed. One of the challenges of T4.4 has been deciding how to represent directed acyclic graphs (DAGs) in a computer understandable format like XML or JSON. The solution we have adopted is BPMN.

BPMN is a standard for business process modeling that provides a graphical notation for specifying business processes.

The definition of new workflows can be made generating a BPMN document, but since this is not user friendly, we have integrated a graphical user interface for the definition of new workflows. Among all the available interfaces we decided to use the **Camunda Modeler** (<https://camunda.com/download/modeler/>) because it is directly usable together with Camunda.

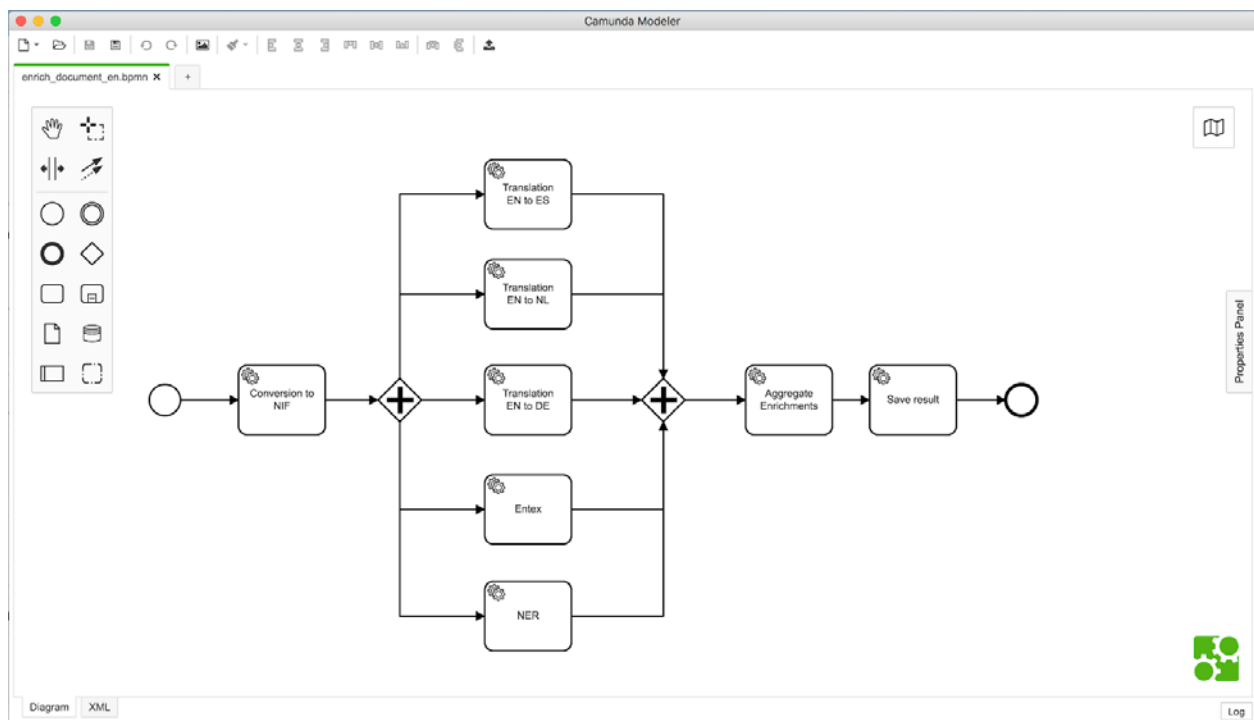


Figure 7. Example of a Workflow defined.

As depicted in Figure 7, the workflows are defined using a drag & drop approach to include new components, each component being one of the services developed in the Lynx project.

3.7 EXECUTION OF A WORKFLOW

This section provides a brief description of the steps needed to test the curation workflow manager. In particular, we use as an example the workflow corresponding to a part of the “LKG population” one, which

has been described in the D4.3 [LynxD43]: the work consists of enriching a document with some of the Lynx WP3 services.

1. Send a POST request using the `/workflows/LKGP` controller. The body of your request must contain a valid LynxDocument JSON. In the parameters, you must specify collection in which you want to store the document and its annotations.
2. Wait until the workflow is completed. You can check the completion percentage using the `/workflows/status/{workflowInstanceId}` controller, the `workflowInstanceId` is provided in the response obtained in step 1.
3. Once the workflow is completed you can verify that the document has been indexed and linked to the LKG:
 - Send a GET request using the Document controller of the DM. The *documentId* should be the one returned in step 1.
 - Perform a search using the Search module API. Search for some words contained in the title of the document ingested in step 1, you should find it in the results.

Since the CWM and the services are under development we cannot ensure that this demo will continue to work in the future. For further details please visit the complete documentation at http://lynx-project.eu/doc/api/alp_wm.html.

4 CONCLUSIONS

This report provides the description of the initial implementation of the Curation Workflow manager and the Document manager. The curation workflow manager implementation described in this report is based on the requirements presented in D1.1 [LynxD11] and D4.1 [LynxD41] and implements the workflows described in D4.3 [LynxD43].

The initial implementation of the curation workflow manager is based on Camunda BPMN Engine together with some self-implemented components, which allow not only the management but also the execution of workflows while the definition of new workflows is done via a graphical interface (Camunda Modeler).

The implementation of the document manager was complicated because at the beginning, it was only regarded as a storage and retrieval module for documents, but it also has the functionality of storing and providing access to the linked data (Legal Knowledge Graph). It has been implemented based on Trellis LDP, and extended with some additional HTTP methods to adapt its functionality to project needs.

Considering that this report is only an initial version of implementation, some modifications are expected both in the DCM and in the WM depending on the development of the pilots. Some steps that must be carried out in the project and are already foreseen are:

- The implementation of the workflows defined in D4.3 that are not yet implemented (*Labour Law* workflow).
- The implementation of some modifications in the curation workflow manager to support more efficient communication capabilities between the services.

ANNEX 1 – IMPLEMENTATION DETAILS (CODE)

This annex contains all the repositories related to DCM and CWM.

Repository URL	Description
https://gitlab.com/superlynx/alp_wm	Workflow Manager engine
https://gitlab.com/superlynx/workers	Workers
https://gitlab.com/superlynx/workflows	Contains the implemented workflows in BPMN format.
https://gitlab.com/superlynx/dcm	Document Manager

Table 1. List of repositories related to DCM and WM.

NOTE: some of the repositories are established as private access but credentials are offered upon request.

Next one finds a report on OpenAPI description of the DCM and CWM' HTTP endpoints. The tool <https://mrin9.github.io/RapiPdf/> was used to generate these reports.

API Reference

Api Documentation

API Version: 1.0

Api Documentation

CONTACT

Terms of service: [urn:tos](#)

INDEX

1. ANNOTATION	3
1.1 GET /collections/{collectionId}/documents/{documentId}/annotations	3
1.2 PUT /collections/{collectionId}/documents/{documentId}/annotations	3
1.3 POST /collections/{collectionId}/documents/{documentId}/annotations	4
1.4 GET /collections/{collectionId}/documents/{documentId}/annotations/**	4
1.5 DELETE /collections/{collectionId}/documents/{documentId}/annotations/**	5
1.6 GET /collections/{collectionId}/documents/{documentId}/annotations/list	6
2. COLLECTIONS	7
2.1 GET /collections	7
2.2 POST /collections/{collectionId}	7
2.3 DELETE /collections/{collectionId}	7
3. DOCUMENTS	9
3.1 GET /collections/{collectionId}/documents	9
3.2 PUT /collections/{collectionId}/documents	10
3.3 POST /collections/{collectionId}/documents	12
3.4 GET /collections/{collectionId}/documents/search	14
3.5 GET /collections/{collectionId}/documents/{documentId}	15
3.6 DELETE /collections/{collectionId}/documents/{documentId}	16

API

1. ANNOTATION

Annotation Controller

1.1 GET /collections/{collectionId}/documents/{documentId}/annotations

Gets all the annotations of a document

Returns a complete representation of the annotations of the document

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId
*documentId	string	documentId

HEADER PARAMETERS

NAME	TYPE	DESCRIPTION
*Accept	string	Accept

RESPONSE

STATUS CODE - 200: OK

RESPONSE MODEL - application/json

string

RESPONSE MODEL - application/json+ld

string

RESPONSE MODEL - application/rdf+xml

string

RESPONSE MODEL - text/turtle

string

RESPONSE MODEL - application/n-triples

string

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

1.2 PUT /collections/{collectionId}/documents/{documentId}/annotations

Updates annotations of a document

This will replace all previous annotations of the document. The request body should be a valid NIF 2.1 document

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId
*documentId	string	documentId

REQUEST BODY - application/rdf+xml

string

RESPONSE

STATUS CODE - 201: Created

STATUS CODE - 204: No Content

RESPONSE MODEL - */*

string

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

1.3 POST /collections/{collectionId}/documents/{documentId}/annotations

Creates new annotation(s)

The request body should be a valid NIF 2.1 document

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId
*documentId	string	documentId

REQUEST BODY - application/rdf+xml

string

RESPONSE

STATUS CODE - 201: Created

RESPONSE MODEL - */*

string

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

1.4 GET /collections/{collectionId}/documents/{documentId}/annotations/

**

Gets an annotation

This will show a specific annotation of a document.

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId
*documentId	string	documentId

HEADER PARAMETERS

NAME	TYPE	DESCRIPTION
*Accept	string	Accept

RESPONSE

STATUS CODE - 200: OK

RESPONSE MODEL - application/json

string

RESPONSE MODEL - application/json+ld

string

RESPONSE MODEL - application/rdf+xml

string

RESPONSE MODEL - text/turtle

string

RESPONSE MODEL - application/n-triples

string

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

**1.5 DELETE /collections/{collectionId}/documents/{documentId}/
annotations/****

Deletes an existing annotation

Deletes an annotation

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId
*documentId	string	documentId

RESPONSE

STATUS CODE - 200: OK

RESPONSE MODEL - */*

string

STATUS CODE - 204: No Content

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

1.6 GET /collections/{collectionId}/documents/{documentId}/annotations/list

Lists all the annotations of a document

Returns an array of annotation identifiers.

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId
*documentId	string	documentId

RESPONSE

STATUS CODE - 200: OK

RESPONSE MODEL - application/json

string

RESPONSE MODEL - application/rdf+xml

string

RESPONSE MODEL - text/turtle

string

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

2. COLLECTIONS

Collection Controller

2.1 GET /collections

Lists the available collections

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: OK

RESPONSE MODEL - */*

string

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

2.2 POST /collections/{collectionId}

Creates a new collection

any slug with no empty spaces

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId

RESPONSE

STATUS CODE - 201: Created

RESPONSE MODEL - */*

string

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

2.3 DELETE /collections/{collectionId}

Deletes an existing collection

the collection must be empty

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId

RESPONSE

STATUS CODE - 202: Accepted

RESPONSE MODEL - */*

string

STATUS CODE - 204: No Content

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

3. DOCUMENTS

Document Controller

3.1 GET /collections/{collectionId}/documents

List documents.

Retrieves a list of all documents in a collection

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
limit	int32	limit
offset	int32	offset
sort	string	sort

RESPONSE

STATUS CODE - 200: OK

RESPONSE MODEL - application/json

```

{
  docs [{
    Array of object:
    @context      string
    annotations [{
      Array of object:
      anchorOf      string
      comment       string
      id            string
      offset_end    integer
      offset_ini    integer
      referenceContext string
      source        string
      taClassRef    string
      taIdentRef    string
      type [{
        Array of object:
      }]
    }]
    id            string
    metadata {
    }
    parts [{
      Array of object:
    }]
  }
}
    
```

```

partsVictor [{
  Array of object:
    @id      string
    @type    string
    offset_end integer
    offset_ini integer
    parent   string
    text     string
    title    string
  }]
text       string
type [{
  Array of object:
}]
}]
total     integer
}
    
```

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

3.2 PUT /collections/{collectionId}/documents

Updates a document

This will update a document. The previous version of the document will be overwritten and the the annotations re-evaluated

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId

REQUEST BODY - application/json

```

{
  @context    string
  annotations [{
    Array of object:
      anchorOf    string
      comment     string
      id          string
      offset_end  integer
      offset_ini  integer
      referenceContext string
      source      string
      taClassRef  string
      taIdentRef  string
      type [{
        Array of object:
      }]
    }]
  id          string
  metadata {
}
    
```

```
parts [{
  Array of object:
}]
partsVictor [{
  Array of object:
    @id      string
    @type    string
    offset_end integer
    offset_ini integer
    parent   string
    text     string
    title    string
}]
text      string
type [{
  Array of object:
}]
}
```

REQUEST BODY - application/rdf+xml

```
{
  @context      string
  annotations [{
    Array of object:
      anchorOf      string
      comment        string
      id             string
      offset_end     integer
      offset_ini     integer
      referenceContext string
      source         string
      taClassRef     string
      taIdentRef     string
      type [{
        Array of object:
      }]
    }]
  id            string
  metadata {
  }
  parts [{
    Array of object:
  }]
  partsVictor [{
    Array of object:
      @id      string
      @type    string
      offset_end integer
      offset_ini integer
      parent   string
      text     string
      title    string
    }]
  text      string
  type [{
    Array of object:
  }]
}
```

RESPONSE

STATUS CODE - 201: Created

STATUS CODE - 204: No Content

RESPONSE MODEL - */*

string

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

3.3 POST /collections/{collectionId}/documents

Creates a new document

If the document has no id, a new one will be assigned. Use JSON for the moment please

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId

REQUEST BODY - application/json

```
{
  @context      string
  annotations [{
    Array of object:
    anchorOf     string
    comment      string
    id           string
    offset_end   integer
    offset_ini   integer
    referenceContext string
    source       string
    taClassRef   string
    taIdentRef   string
    type [{
      Array of object:
    }]
  }]
  id            string
  metadata {
  }
  parts [{
    Array of object:
  }]
  partsVector [{
    Array of object:
    @id         string
    @type       string
    offset_end  integer
    offset_ini  integer
  }]
}
```

```
    parent    string
    text      string
    title     string
  }}
  text       string
  type [ {
    Array of object:
  } ]
}
```

REQUEST BODY - application/rdf+xml

```
{
  @context   string
  annotations [ {
    Array of object:
      anchorOf    string
      comment     string
      id           string
      offset_end  integer
      offset_ini  integer
      referenceContext string
      source      string
      taClassRef  string
      taIdentRef  string
      type [ {
        Array of object:
      } ]
    } ]
  id         string
  metadata {
  }
  parts [ {
    Array of object:
  } ]
  partsVictor [ {
    Array of object:
      @id        string
      @type      string
      offset_end integer
      offset_ini integer
      parent     string
      text       string
      title     string
    } ]
  text       string
  type [ {
    Array of object:
  } ]
}
```

RESPONSE**STATUS CODE - 201: Created****RESPONSE MODEL - */***

string

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

3.4 GET /collections/{collectionId}/documents/search

Makes a search

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
limit	int32	limit
offset	int32	offset
query	array of string	query

RESPONSE

STATUS CODE - 200: OK

RESPONSE MODEL - application/json

```

{
  docs [{
    Array of object:
      @context      string
      annotations [{
        Array of object:
          anchorOf      string
          comment        string
          id              string
          offset_end     integer
          offset_ini     integer
          referenceContext string
          source          string
          taClassRef     string
          taIdentRef     string
          type [{
            Array of object:
          }]
        }]
      id              string
      metadata {
      }
      parts [{
        Array of object:
      }]
      partsVictor [{
        Array of object:
          @id          string
    
```

```

        @type      string
        offset_end integer
        offset_ini integer
        parent     string
        text       string
        title      string
    }
    text          string
    type [{
    Array of object:
    }]
    }
    total        integer
}

```

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

3.5 GET /collections/{collectionId}/documents/{documentId}

Retrieves an existing document

The document has not annotations, in principle

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId
*documentId	string	documentId

HEADER PARAMETERS

NAME	TYPE	DESCRIPTION
*Accept	string	Accept

RESPONSE

STATUS CODE - 200: OK

RESPONSE MODEL - application/json

string

RESPONSE MODEL - application/json+ld

string

RESPONSE MODEL - application/rdf+xml

string

RESPONSE MODEL - text/turtle

string

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

STATUS CODE - 404: Not Found

3.6 DELETE /collections/{collectionId}/documents/{documentId}

Deletes an existing document

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*collectionId	string	collectionId
*documentId	string	documentId

RESPONSE

STATUS CODE - 200: OK

STATUS CODE - 204: No Content

STATUS CODE - 401: Unauthorized

STATUS CODE - 403: Forbidden

API Reference

Workflow manager

API Version: 0.0.0

Responsible for the effective orchestration of the microservices for the execution of workflows. Workflows are combinations of both parallel and sequential tasks and are specified using Directed Acyclic Graphs.

CONTACT

NAME: Lynx-Project API Team

EMAIL: apiteam@lynx-project.eu

URL: <https://lynx-project.eu>

Terms of service: <http://lynx-project.eu/terms/>

INDEX

1. WORKFLOWS EXECUTION	3
1.1 POST <code>/workflows/LKGP</code>	3
2. WORKFLOWS STATUS	4
2.1 GET <code>/workflows/status/{workflowInstanceId}</code>	4

API

1. WORKFLOWS EXECUTION

1.1 POST /workflows/LKGP

Start the ingestion of a document in the Lynx system

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
collectionId	string	The LKG collection in which the enriched document will be stored.

REQUEST BODY - application/json

```
{
  text      string
  metadata {
    language* string
  }
  parts [{
    Array of object:
    offset_ini* string
    offset_end* string
    title*      string
  }]
}
```

RESPONSE

2. WORKFLOWS STATUS

2.1 GET /workflows/status/{workflowInstanceId}

Start the ingestion of a document in the Lynx system

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
workflowInstanceId	string	The LKG collection in which the enriched document will be stored.

RESPONSE

STATUS CODE - 200: The completion percentage

RESPONSE MODEL - text/plain

string

REFERENCES

- [OMG2011] OMG. (2011). Business Process Model and Notation (BPMN), Version 2.0, January.
- [LynxD11] Jorge González-Conejero, Emma Teodoro, & Pompeu Casanovas. (2018). Lynx D1.1 Functional Requirements Analysis Report. <https://zenodo.org/record/1256836>
- [LynxD24] Víctor Rodríguez-Doncel, Socorro Bernardos, Rebeca Varela, Patricia Martín. (2018). Lynx D2.4 Data Management Plan. <https://zenodo.org/record/3236320>
- [LynxD41] Julián Moreno-Schneider & Georg Rehm. (2018). D4.1 Pilots Requirements Analysis Report. <https://zenodo.org/record/3236427>
- [LynxD42] Julián Moreno-Schneider & Georg Rehm. (2018). D4.2 Intermediate version of Workflow definition. <https://zenodo.org/record/1745324>
- [LynxD43] Julián Moreno-Schneider & Georg Rehm. (2019). D4.3 Final version of Workflow definition. <https://doi.org/10.5281/zenodo.3235767>
- [Bourgonje2016] Peter Bourgonje, Julián Moreno Schneider, Georg Rehm, and Felix Sasaki. Processing Document Collections to Automatically Extract Linked Data: Semantic Storytelling Technologies for Smart Curation Workflows. In Aldo Gangemi and Claire Gardent, editors, *Proceedings of the 2nd International Workshop on Natural Language Generation and the Semantic Web (WebNLG 2016)*, pages 13-16, Edinburgh, UK, 9 2016. The Association for Computational Linguistics.
- [MorenoSchneider2018a] Julian Moreno Schneider & Georg Rehm. Towards a Workflow Manager for Curation Technologies in the Legal Domain. In Georg Rehm, Víctor Rodríguez-Doncel, and Julian Moreno Schneider, editors, *Proceedings of the LREC 2018 Workshop on Language Resources and Technologies for the Legal Knowledge Graph*, pages 30-35, Miyazaki, Japan, 5 2018. 12 May 2018.
- [MorenoSchneider2018b] Julian Moreno Schneider and Georg Rehm. Curation Technologies for the Construction and Utilisation of Legal Knowledge Graphs. In Georg Rehm, Víctor Rodríguez-Doncel, and Julian Moreno Schneider, editors, *Proceedings of the LREC 2018 Workshop on Language Resources and Technologies for the Legal Knowledge Graph*, pages 23-29, Miyazaki, Japan, 5 2018. 12 May 2018.