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

## D1.4 Setup and implementation of the basic platform

| | |
|---|---|
| **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** | Filippo Maganza (ALP), Kennedy Junior Anagbo (ALP) |
| **CONTRIBUTORS** | Sotiris Karampatakis (SWC), Víctor Rodríguez-Doncel (UPM) |
| **REVIEWERS** | (SWC), (DFKI) |
| **VERSION \| STATUS** | V1.0 \|Final |
| **NATURE** | Report |
| **DISSEMINATION LEVEL** | Public |
| **DOCUMENT DOI** | 10.5281/zenodo.3236427 |
| **DATE** | 31/05/2019 (M18) |

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 draft | 23/03/2019 | Filippo Maganza (ALP), Kennedy Junior Anagbo (ALP) |
| 0.2 | Document structure | 25/03/2019 | Filippo Maganza (ALP), Kennedy Junior Anagbo (ALP) |
| 0.3 | Added Introduction | 01/04/2019 | Filippo Maganza /(ALP), Kennedy Junior Anagbo (ALP) |
| 0.4 | Added Chapter 2 | 15/04/2019 | Filippo Maganza (ALP), Kennedy Junior Anagbo (ALP) |
| 0.5 | Added Chapter 3 | 02/05/2019 | Filippo Maganza (ALP), Kennedy Junior Anagbo (ALP), Sotiris Karampatakis (SWC), Victor Rodriguez Doncel (UPM) |
| 0.6 | Added Chapter 4 | 10/05/2019 | Filippo Maganza (ALP), Kennedy Junior Anagbo (ALP) |
| 0.7 | Added Executive Summary | 10/05/2019 | Filippo Maganza (ALP), Kennedy Junior Anagbo |
| 0.8 | Integration of reviewers' comments | 24/05/2019 | Filippo Maganza (ALP), Kennedy Junior Anagbo (ALP) |
| 0.9 | Integration of other contributions from Lynx partners | 27/05/2019 | Filippo Maganza (ALP), Kennedy Junior Anagbo (ALP) |
| 0.99 | Final remarks | 30/05/2019 | Filippo Maganza (ALP), Kennedy Junior Anagbo (ALP) |
| 1.0 | New Test Portal Subsection | 31/05/2019 | Víctor Rodríguez-Doncel (UPM) |

## ACRONYMS LIST

| | |
|---|---|
| API | Application Programming Interface |
| CRUD | Create Read Update Delete |
| HTTP | Hypertext Transfer Protocol |
| IoC | Inversion of Control |
| JDBC | Java Database Connectivity |
| JDK | Java Development Kit |
| JMS | Java Message Service |
| JMX | Java Management Extensions |
| JSON | JavaScript Object Notation |
| JWT | JSON Web Token |
| LKG | Legal Knowledge Graph |
| MVC | Model-View-Controller |
| PaaS | Platform as a Service |
| RDF | Resource Description Framework |
| REST | Representational State Transfer |
| SaaS | Software as a Service |
| SOAP | Simple Object Access Protocol |
| WAR | Web Application Resource |
| XML | Extensible Markup Language |

## EXECUTIVE SUMMARY

The transformation of the high-level architectural design of D1.3 “Technical architecture design report” into basic setup and implementation of the Lynx platform is the purpose of this deliverable. The basic implementation of the various foundational microservices that compose the basic platform and the frameworks used in realising these implementations are amongst the topics presented in this report.

The basic implementation of these structural components is subject to future modifications and therefore the basic platform setup and implementation of the Lynx platform is projected to evolve over time.

The choice of the Spring Framework is motivated by the requirements (D1.1 “Functional requirements analysis report”, D1.2 “Technical requirements analysis report”, and D4.1 “Pilots requirements analysis report”) that the Lynx platform addresses; the framework adapts very well to meeting the realisation of these requirements.

## TABLE OF CONTENTS

## TABLE OF FIGURES

# 1 INTRODUCTION

In D1.3, “Technical architecture design report”, a blueprint for the Lynx system relating to the technical architecture design was presented with detailed descriptions of the major structural components that make up the design solution; the dependencies between them, their responsibilities and how they work together. In other words, a high-level technical architectural design of the system was presented resulting from the combination of the requirements documented in deliverables D1.1 “Functional requirements analysis report”, D1.2 “Technical requirements analysis report”, and D4.1 “Pilots requirements analysis report”.

The major structural components of the platform which have been identified as foundational microservices as documented in D1.3 “Technical architecture design report” constitute the foundation of the Lynx platform. Additionally, other peripheral microservices depend upon this basic platform. The foundational microservices that have been identified and defined are:

- API Manager (also referred to as API)
- Authentication and Identity Manager (OAuth)
- Document Manager (DCM)
- Workflow Manager (WM)

Amongst these structural components, the workflow manager shall be presented in WP4 whilst the rest are described in this document.

## 1.1 PURPOSE AND STRUCTURE OF THIS DOCUMENT

This document provides a detailed description of the basic implementation of the Lynx platform with motivation derived from the design decisions presented in D1.3.

This document presents implementation details of the foundational microservices as described in D1.3.

The rest of this document is organized as follows. Section 2 presents and describes the implementation frameworks and programming languages used for the basic implementation of the Lynx platform. Section 3 describes the basic implementation of the foundational microservices detailing out important implementation characteristics. Section 4 summarises the deliverable and presents future work.

# 2 IMPLEMENTATION FRAMEWORKS

The Lynx microservice architecture has been designed to support different implementation frameworks and/or programming languages. The main motivation for this design choice is that the Lynx project involves a large number of different teams with different skills and technical backgrounds thus, we decided to emphasize those skills by adding a level of technical abstraction between the architecture components since microservice architecture provides better testability, service interoperability, improved scalability and increased flexibility; characteristics that adapt well to the Lynx distributed service nature.

In the following sections, we present a general description of the frameworks that have been adopted for the implementation of the Lynx basic platform.

## 2.1 SPRING FRAMEWORK

Spring [1] is an open source application framework that provides comprehensive programming and configuration models. Although the framework does not impose any specific programming language, it has become popular in the Java community. Spring provides an infrastructural support at the application level.

The Spring framework includes several modules that provide a range of services. The ones mostly used by Lynx are:

- ***Spring Core Container***: it is the heart of the Spring framework and all other modules are built on it. It provides the dependency injection feature also known as inversion of control (IoC).
- ***Spring Security***: it is an authentication and access control framework which is highly customizable used to secure spring-based applications.
- ***Spring Data***: it provides an easy access to several data stores while maintaining the data store's unique traits by significantly reducing the amount of code required to implement data access layers for several persistence stores.

### 2.1.1 Spring Security Framework

Spring security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

It is a framework that focuses on providing both authentication and access control to applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements [2].

Spring security framework has an implementation for OAuth 2.0 which we have adopted for the Lynx platform. A detailed description of OAuth (formely named AUTH) is documented in D1.3 section 4.1.2.

## 2.2 SPRING BOOT

Spring Boot is a project built on top of the Spring framework. It provides a simpler and faster way of setup, configure, and run both simple and web-based applications. It makes it easy to create stand-alone, production-grade Spring based Applications.

The following are the features of Spring Boot that can be used:

- Embed Tomcat or Jetty directly (no need to deploy WAR files)
- Provide opinionated "starter" dependencies to simplify build configuration

[1] https://spring.io/

- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks and externalized configuration
- No code generation and no requirement for XML configuration

# 3 BASIC PLATFORM IMPLEMENTATION

The basic platform consists of the foundational microservices we defined in D1.3 “Technical architecture design report” on which the other services rely. These components are:

- API Manager (API)
- Authentication and Identity Manager (OAuth)
- Document Manager (DCM)
- Workflow Manager (WM, to be described in D4.4)

In this section, we provide a detailed description of the API, OAuth, and the DCM specifying their responsibilities, architecture and some implementation details.

Although in this section we present the realization of the basic platform, it is important to specify that the implementations are subject to change since we envision an evolution of the foundational microservices.

Currently, the implementation of all the components make use of the Spring Boot project of the Spring framework. Implementation of the various components has been accomplished using Java. A demo

## 3.1 API MANAGER (APIM)

The API manager microservice, as described in D1.3, is an API gateway that performs routing of the incoming (external) requests to the correct microservices and enforcing throttling and security policies.

The main responsibilities of the Lynx API manager are:

- To provide a single point of access to the platform
- To provide an additional layer of abstraction between the client applications and the microservices
- The very first level of access control
- The throttling of the incoming requests in order to mitigate overwhelming the platform with too many requests within a specified time interval and against possible Denial of Service attacks.

The implementation of the routing of external requests was achieved using the Netflix’s API gateway called Zuul. Zuul[2] is built to enable dynamic routing, monitoring, resiliency and security. Lynx has also implemented rate limiting using an open source library[3]. The implementation limits the number of calls that can be made to a service within a certain refresh interval window to a predefined number by setting the limit property of the application.

If this predefined number is exhausted before a refresh time interval expires, the next request is rejected until the next refresh interval. The refresh property is also configurable.

A Redis[4] database is used for storing the data of the rate limit implementation. The Spring framework modules used in implementing this foundational microservice are:

- Spring Core Container
- Spring Security OAuth 2.0
- Spring Data Redis

[2] https://github.com/Netflix/zuul

[3] https://github.com/marcosbarbero/spring-cloud-zuul-ratelimit

[4] https://redis.io/

## 3.2 OAUTH AUTHORIZATION SERVER AND IDENTITY MANAGER (OAUTH)

The OAuth foundational microservice manages the users and client applications identities and the associated information and supports the authorization server functionalities for the OAuth 2.0 protocol. Currently, the supported OAuth grant types are "*client credentials*" and "*password*".

The Spring framework modules used in implementing this foundational microservice are:

- Spring Core Container
- Spring Security OAuth 2.0
- Spring Security JWT
- Spring Data MongoDB

### 3.2.1 Client Application and User classes

The Client Application and User classes we have defined represent the data identities of the entities that can access protected resources of the platform.

A Client Application represents the identity of a trusted client application of the Lynx platform; for example, the applications of Openlaws, Cuatrecasas and DNV GL.

A User represents the identity of a Lynx user. Currently, Lynx users are only administrators and developers of the platform with the sole responsibility of testing and managing the platform. In the future, we envision the possibility of having also Lynx end users.

### 3.2.2 REST API Layer

In order to manage our client applications and users, some RESTful API endpoints have been defined that provide CRUD operations over their identities.

The RESTful APIs provide methods for the following operations:

- Retrieving the list of client applications or users
- Retrieving a single client application or user
- Creating a new client application or user
- Updating an existing application or user
- Deleting an existing client application or user

The full OpenAPI 3 specification of the RESTful APIs of the OAuth service are published and accessible online[5].

Both the API manager and the OAuth microservices are running and have been successfully deployed to an OpenShift instance. Figure 1 shows the deployment of the API and OAuth microservices.

[5] http://lynx-project.eu/doc/api/yamls/auth.yaml

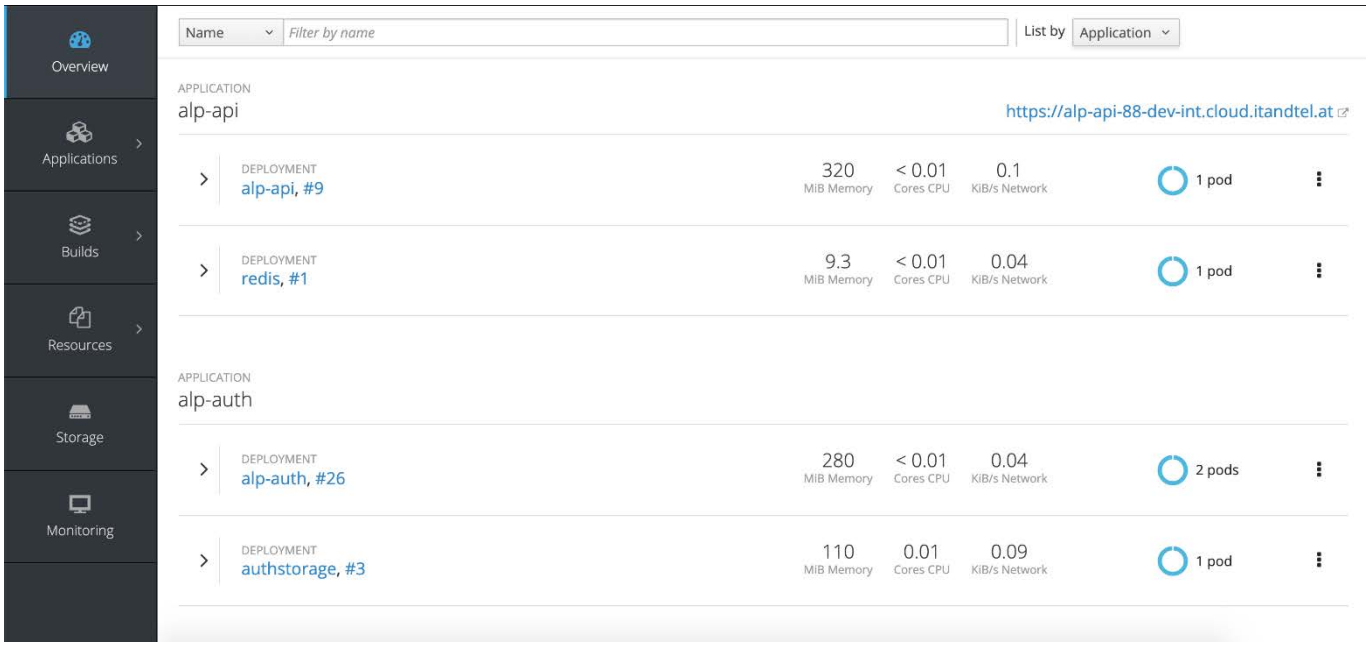


Figure 1 Overview of an OpenShift instance showing deployed API and OAuth microservices

### 3.2.3 Authentication and access token granting

The term authentication describes the process of validating an entity's (client application or user) credentials to verify its identity whereas authorization deals with the definition of policies that determines the resource(s) an entity is permitted to access after a successful authentication process.

Regarding the authorization phase, the Lynx consortium will be responsible to defining the policies that will govern the use of the Lynx resources.

The OAuth framework defines a group of grant types that can be used to get an access token. The grant types that will be used in the Lynx platform are:

- ***Client Credentials***: the "client credentials" grant is used when applications request an access token to access their own resources, not on behalf of a user.
- ***Password grant***: the "password" grant type is used by first-party clients to exchange a user's credentials for an access token.

A brief summary of the token grant flows that have been implemented is presented as follows:

**Client credentials grant**

1. A client application requests an access token by authenticating with the Authorization Server using its attributes.
2. Provided that the client application is successfully authenticated, the Authorization Server issues a JWT representing the identity of the client application and sends it back to it.

**Password grant**

1. A client application requests an access token on the behalf of a user by authenticating with the Authorization Server using its attributes and the user's credentials.
2. Provided that the client application and the user are successfully authenticated, the Authorization Server issues a JWT representing the identity of the user and sends it back to the client.

The full specification of client credentials grant can be found at RFC 6749 [2] Section 4.4, whereas the password grant method is specified by RFC 6749 Section 4.3.

### 3.2.4 Requests to protected resources and access control

After a client application (or a user) has been fully authenticated and authorized, it can then access protected resources of the Lynx platform. The following points summarize the flow for requesting an access to a Lynx protected resource:

1. The client application (or user) requests access to a protected resource by a Resource Server and authenticates by presenting the JWT.
2. The Resource Server verifies the signature of the JWT and the client application's or user's authorities against the one needed to access the resource.
3. Provided that the JWT is valid and the authorities are sufficient to access the resource, the Resource Server serves the Client Application's request.

### 3.2.5 Persistence layer

For the implementation of the persistence layer, we decided to serialize the clients' applications and users' identities in JSON format and save them in a MongoDB database.

## 3.3 DOCUMENT MANAGER (DCM)

The Document manager (also referred to as *DCM*) forms a central part of the Lynx platform in terms of the general platform functioning capabilities; this is where the documents are stored and maintained. Its basic 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 contain mentions of this entity"), 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 CRUD APIs to manage the following resources within the Lynx platform: documents and collections.

The basic entities managed by the DCM are described in detail in D2.4 "Data Management Plan" (See the Data Models section): *collections, documents and annotations*.

- A ***document*** is a piece of information in plain text, RDF or JSON format which contains structured text, metadata and annotations. The DCM hides the way documents are stored, but they can be retrieved as RDF or JSON. Original documents such as PDFs are not stored in the LKG.
- A ***collection*** represents a group of documents with an associated label or in a different logical store.
- An ***annotation*** is any piece of information that was not included in the original document but has been added by the Lynx services.

Documents are identified as described in the URI minting section of D2.4 "Data Management Plan" where the data structures for documents, collections and annotations are also defined. Documents are represented either as a simple JSON or as RDF (which in turns permits storing documents as JSON-LD).

### 3.3.1 Architecture

The architecture of the implemented DCM can be depicted in three distinct layers which are the following:

- REST Controllers layer
- Data Layer
- Backend Layer

Additionally, to these layers, a common libraries package has been developed in order to define data models to be used on the DCM and possibly on other Lynx Services.

An abstract overview of the architecture can be seen on Figure 2.

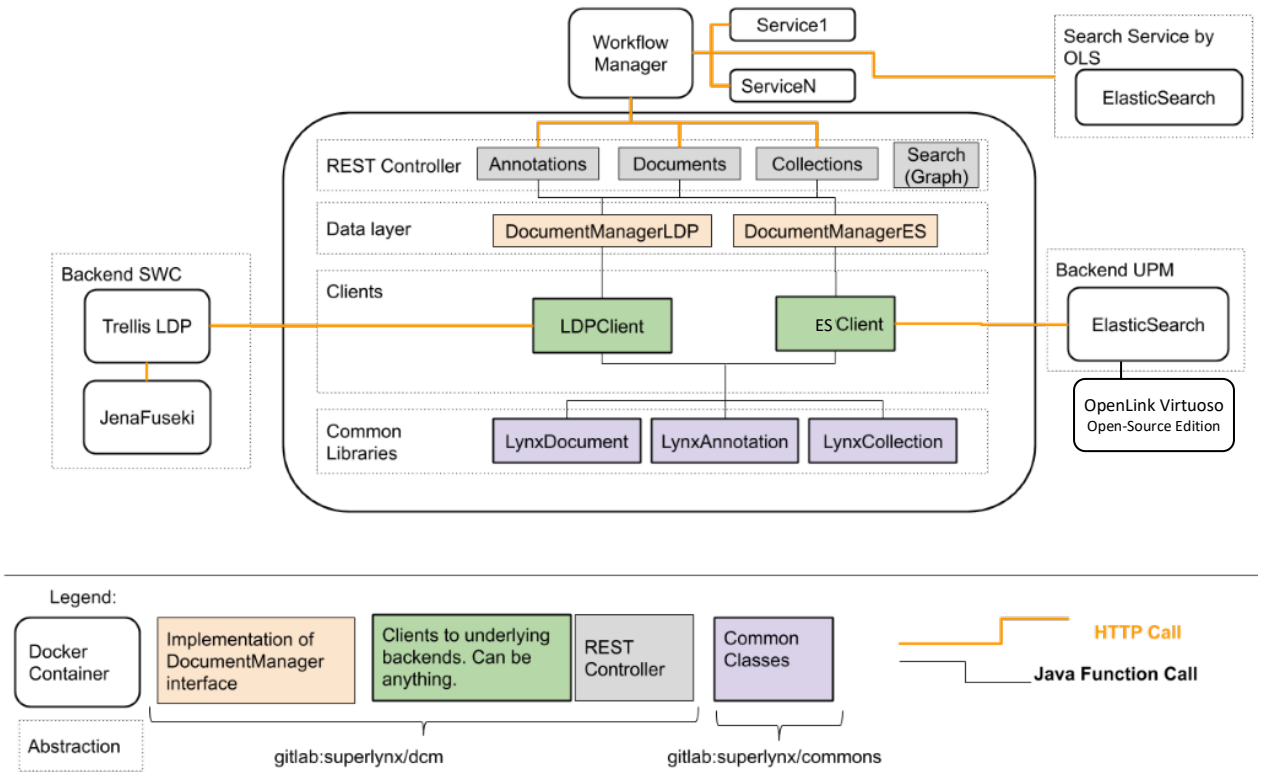


Figure 2 Abstract Overview of the DCM Architecture

### 3.3.2 REST API Layer

In the conceptualization of the Document manager, a collection is a unique group, that contains unique documents, and each document may contain a set of annotations produced by the various services offered by the Lynx Platform. These abstractions are reflected on the common libraries we have defined; the LynxCollection, the LynxDocument and the LynxAnnotation. The REST API provides a set of CRUD operations (controllers) over these models, one distinct for each model; the CollectionsController, the DocumentsController and the AnnotationsController. An additional Search controller is provided; to be used in order to perform SPARQL queries on the documents.

The basic implementation of this microservice provides controller methods for the following operations:

**Collections Controller**

- A get method for retrieving all available collections
- A post method for creating a new collection
- A delete method for deleting an existing collection

**Document Controller**

- Get methods for retrieving a list of documents and an existing document text
- A post method for creating a new document

- A put method for performing updates on a document
- A delete method for deleting an existing document

**Annotations Controller**

- Get methods for retrieving all the annotations of a document and single annotations
- A post method for creating new annotations
- A put method for updating annotations of a document
- Delete methods for deleting all annotations of a document or an existing annotation

The Spring Framework is used to implement the REST API layer of the DCM. For all the controllers, the Springfox library is used to provide an OpenAPI compatible specification. The Springfox library, based on Swagger, allows the automated generation of machine and human readable specification for the API, by examining the application at runtime, to infer API semantics based on the configurations, the structure and other compile-time Annotations. The specification is then used to provide a UI for testing purposes.

### 3.3.3 Data Layer

The Document manager Interface defines abstractly all the possible methods to perform CRUD operations over the models. The REST API uses this interface to call the different methods, allowing to seamlessly switch over implementations. The active implementation to be used is configured on the runtime; an environment variable has to be defined in the deployment environment of the DCM. For the testing purposes, both solutions have been deployed on the OpenShift cluster of the Lynx Project.

On the Data Layer of the DCM, the implementations of the Document manager Interface are defined. Currently, two Implementations of the Document manager Interface exist, one using the LDP[6] Trellis[7] as a backend and the other using an EleasticSearch[8] node.

### 3.3.4 Backend Layer

Currently, two different implementations have been developed for the backend layer; one based on Trellis Linked Data Platform (LDP) and the other based on ElasticSearch. The motivation for the two different implementations stems from the fact that it is challenging to predict at this stage of the project which solution will comply and will be well adapted to the required needs of the platform. Thus, we have opted for these two solutions pending a future concrete decision on one of them.

#### 3.3.4.1 *Trellis LDP*

The DocumentManagerLDP implementation of the Document manager Interface uses the Trellis LDP platform to store the actual documents and their metadata (annotations). An LDP server such as Trellis allows the creation and manipulation of web resources in a Linked Data environment. Trellis is coupled with a triplestore; in our test environment, we use the Jena Fuseki triplestore, in order to store RDF resources or the description of non-RDF resources. Thus, complex queries using the expressiveness of SPARQL, the standard query language for RDF data, can be evaluated over the data.

We developed a Java client according to the LDP Specification to add an additional abstraction layer between the implementation and the actual backend. The LDPclient is then responsible to create HTTP

6 https://dvcs.w3.org/hg/ldpwg/raw-file/default/ldp.html

7 https://www.trellisldp.org/

8 https://www.elastic.co/

requests towards the LDP server. This allows to use other LDP server implementation in case Trellis fail to fulfill the requirements of the platform.

#### 3.3.4.2 *ElasticSearch*

The DocumentManagerES implementation of the Document manager Interface uses an ElasticSearch node to store the documents and their annotations. The endpoint ElasticSearch node is detached from the Document manager, e.g. both systems possibly live in different servers, enabling the easy replacement of stores.

In addition, a Virtuoso RDF Triple Store (the Community Edition of this OpenLink product) has been set in place with a dump of the LKG store enabling SPARQL queries. This store is currently available at the following endpoint and will be regularly updated:

`http://sparql.lynx-project.eu`

The SPARQL endpoint is not meant for UPDATE queries and the offered data being delayed (e.g., regular updates are foreseen). The Document Manager has been deployed to an OpenShift instance hosted at Itandtel.at, as can be seen in Figure 3.

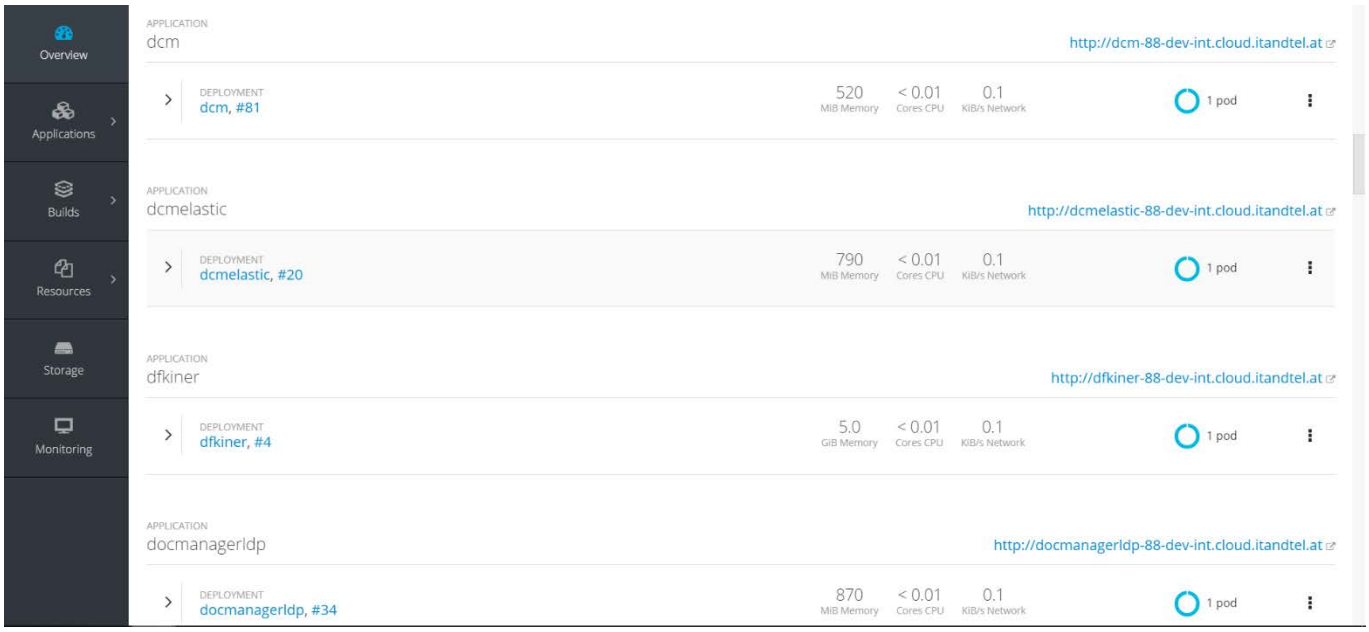


Figure 3 An Overview of the deployment of the Document manager to an OpenShift instance

### 3.4 TEST PORTAL (PORTALT)

A test portal, used for developing and testing different services and components of Lynx, has also been published and it is accesible under the following URI:

`http://lkg.lynx-project.eu/`

This portal uses the DCM to access a LKG and provides a basic search and display of documents. The portal shows two pages; the first one is the search page and short results, the second displays the documents. The search page (Figure 4) consists of a free search text box, plus three combo boxes to filter by document type, jurisdiction and language. As of M18, the portal was handling about 10,000 documents, mostly extracted from Spanish legislation but also including the labor law-specific documents such as collective agreements etc.


Figure 4. Search page of the Test Portal

The second page (see screenshot in Figure 4) simply displays the document along with its metadata records (type, validity, language, jurisdiction, keywords). The document is accessible in both JSON and RDF, as well as with a link to the original PDF file. A text version is also available. There is a pattern to retrieving these documents:

```
http://lkg.lynx-project.eu/res/BOE-A-1997-12735
http://lkg.lynx-project.eu/res/BOE-A-1997-12735.rdf
http://lkg.lynx-project.eu/res/BOE-A-1997-12735.json
http://lkg.lynx-project.eu/res/BOE-A-1997-12735.txt
```

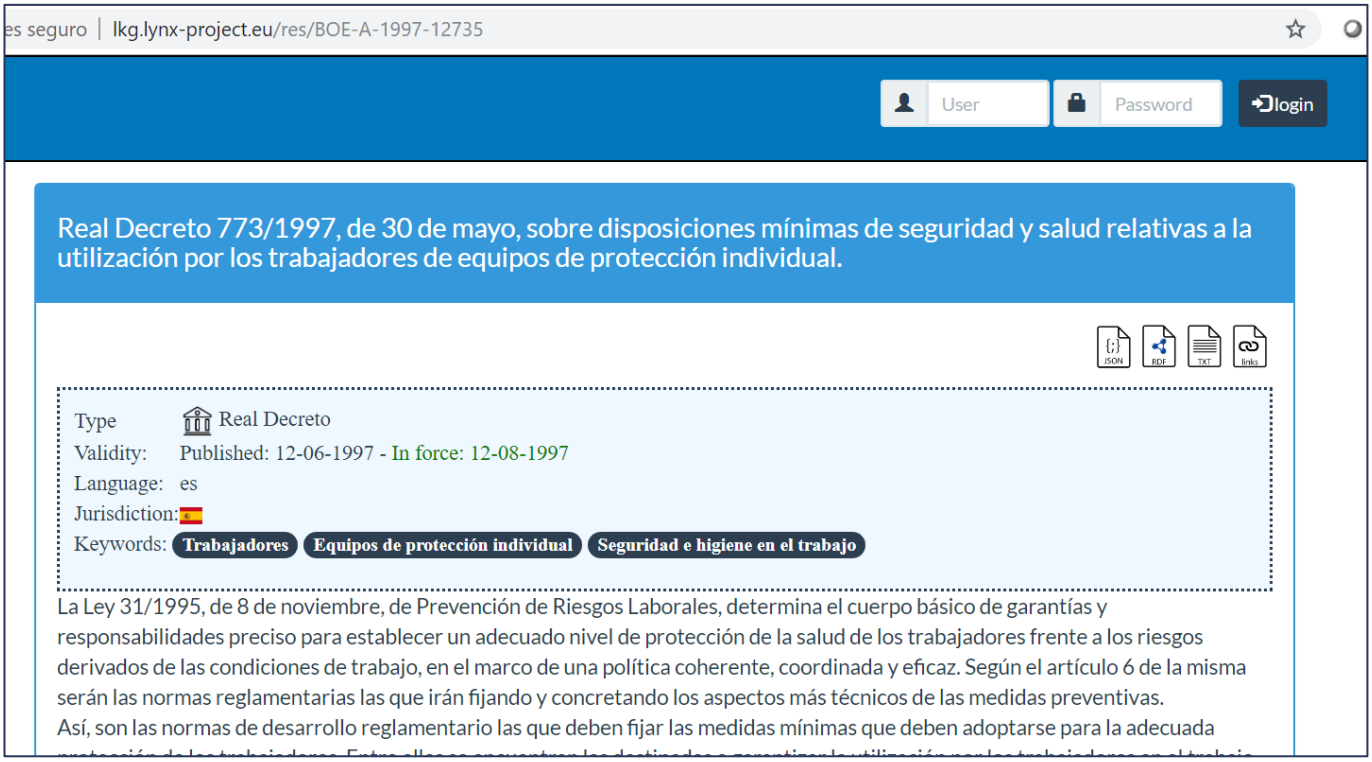


Figure 5. Document page of the Test Portal

The document is displayed with some minimal structuring information and having some minimal linking (to other documents in the LKG). After the login and password have been introduced, a set of new functionalities is made available, as depicted in Figure 6.


Figure 6. Advanced funcionalities for a Lynx Document

Figure 6 shows a new bar of commands, with buttons to go home (home), delete de document (delete) or create a new document (new), as shown in Figure 7. The interface also permits annotating temporal expressions using the TimEx service (time), or to make other annotations using Dbpedia Spotlight (annotate). The document can also invoke the Trans service to do machine translation (ES->EN) and finally, it can offer recommendations based on the topic modelling algorithms of UPM'S LibrAIry[9]. Besides, there is an administration page and a logout button. This software has been made for testing purposes and may disappear as long as the other interfaces appear.

The software is available in the Lynx software repo at:

`https://gitlab.com/superlynx/upm/tree/master/lynx-tools/portal`

[9] http://librairy.github.io/

# Creation of resources

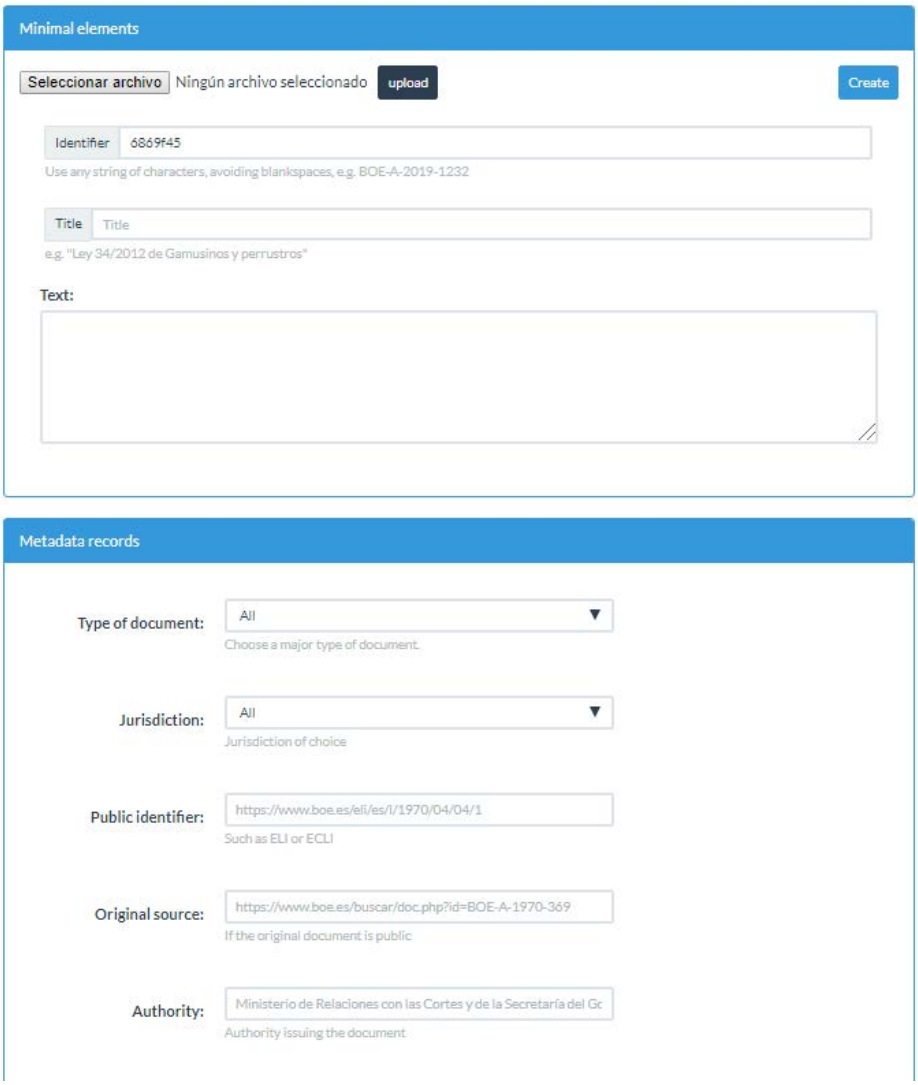


**Figure 7. Creation of new resources from the Test Portal (PortalT)**

# 4 CONCLUSION AND FUTURE WORK

The deliverable presents the basic implementation of the Lynx platform. It describes the basic implementation of the structural components of the platform providing the frameworks that aid in the individual implementation.

The basic platform is envisioned to evolve over time as a direct consequence of the changes that could take place in the basic implementation of these structural components in the future; as such, this report is strictly to be considered as a preliminary update of the basic platform implementation. Nevertheless, the technological evolution of the basic platform should not affect in any way the other peripheral services.

The integration of the basic platform with the peripheral microservices and the realization of the first prototype of the Lynx platform is ongoing in T3.5 “Services/platform integration” and is expected to end in M24.

# REFERENCES

[1] P. Software, “Spring Security,” [Online]. Available: https://spring.io/projects/spring-security. [Accessed 2 April 2019].

[2] I. E. T. F. (IETF), “The OAuth 2.0 Authorization Framework,” [Online]. Available: https://tools.ietf.org/html/rfc6749.

[3] P. Software, “Spring Boot,” [Online]. Available: https://spring.io/projects/spring-boot. [Accessed 28 March 2019].