EUROPEAN LANGUAGE GRID

D2.2

Specification of the ELG platform architecture

Authors:	Stelios Piperidis (ILSP), Dimitris Galanis (ILSP), Miltos Deligiannis (ILSP), Katerina Gkirtzou (ILSP), Penny Labropoulou (ILSP), Georg Rehm (DFKI), Florian Kintzel (DFKI), Maria Moritz (DFKI), Ian Roberts (USFD), Kalina Bontcheva (USFD)
Dissemination Level:	Public
Date:	30-06-2019

About this document

Project	ELG – European Language Grid
Grant agreement no.	825627 – Horizon 2020, ICT 2018-2020 – Innovation Action
Coordinator	Dr. Georg Rehm (DFKI)
Start date, duration	01-01-2019, 36 months
Deliverable number	D2.2
Deliverable title	Specification of the ELG platform architecture
Type	Report
Number of pages	36
Status and version	Final – Version 1.0
Dissemination level	Public
Date of delivery	Contractual: 30-06-2019 – Actual: 29-06-2019
WP number and title	WP2: Grid Platform – Language Grid
Task number and title	Task 2.1: Req. collection and specific. of the higher-level ELG architecture
Authors	Stelios Piperidis (ILSP), Dimitris Galanis (ILSP), Miltos Deligiannis (ILSP), Katerina Gkirtzou (ILSP), Penny Labropoulou (ILSP), Georg Rehm (DFKI), Florian Kintzel (DFKI), Maria Moritz (DFKI), Ian Roberts (USFD), Kalina Bontcheva (USFD)
Reviewers	Khalid Choukri (ELDA), Katrin Marheinecke (DFKI)
Consortium	Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
	Institute for Language and Speech Processing (ILSP), Greece
	University of Sheffield (USFD), United Kingdom
	Charles University (CUNI), Czech Republic
	Evaluations and Language Resources Distribution Agency (ELDA), France
	Tilde SIA (TILDE), Latvia
	Sail Labs Technology GmbH (SAIL), Austria
	Expert System Iberia SL (EXPSYS), Spain
	University of Edinburgh (UEDIN), United Kingdom
EC project officers	Philippe Gelin, Alexandru Ceausu
For copies of reports and other ELG-related information, please contact:	DFKI GmbH European Language Grid (ELG) Alt-Moabit 91c D-10559 Berlin Germany Dr. Georg Rehm, DFKI GmbH georg.rehm@dfki.de Phone: +49 (0)30 23895-1833 Fax: +49 (0)30 23895-1810 http://european-language-grid.eu © 2019 ELG Consortium

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

ACID	Atomicity, Consistency, Isolation, Durability
AI	Artificial Intelligence
API	Application Programming Interface
ASR	Automatic Speech Recognition
CD	Continuous Delivery/Deployment
CEF	Connecting Europe Facility
CI	Continuous Integration
CMDI	Component MetaData Infrastructure
CMS	Content Management System
CORS	Cross-Origin Resource Sharing
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
DSF	Django Software Foundation
DoS	Denial of Service
ELG	European Language Grid
FAIR	Findable, Accessible, Interoperable, Reusable
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IdP	Identity Provider

IE	Information Extraction
JSON	JavaScript Object Notation
JWT	JSON Web Token
LR	Language Resource
LRT	Language Resource and Technology
LT	Language Technology
MQ	Message Queue
MT	Machine Translation
MVP	Minimum Viable Product
NCC	National Competence Center
OAI-PMH	Open Archives Initiative Protocol for Metadata Harvesting
ORM	Object Relational Mapper
OWL	Ontology Web Language
POC	Proof of Concept
RDBMS	Relational Database Management System
RDF	Resource Description Framework
REST	Representational State Transfer
RFC	Request for Comments
SME	Small Medium Enterprise
SSO	Single Sign-On
SQL	Structured Query Language
TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
XML	eXtensible Markup Language
YAML	YAML Ain't Markup Language

List of Terms

Content	Any type of assets included in the ELG platform, including functional and non-functional content, as defined in the list of terms
Functional content	Language processing tools and services that can be executed either locally or at a cloud infrastructure
Language Resource	A resource composed of linguistic material used in the construction, improvement and/or evaluation of language processing applications, but also, in a broader

	sense, in language and language-mediated research studies and applications; examples include datasets of various types, such as textual, multimodal/multimedia corpora, lexical data, grammars, language models, etc. in machine readable form. The term is often used in the bibliography and related initiatives with a broader meaning, encompassing also the (a) tools/ services used for the processing and management of datasets, and (b) standards, guidelines and similar documents that support research, development and evaluation of Language Technology (cf. http://languagelog.ldc.upenn.edu/myl/ldc/LR_background.html). In this document, and in order to easily distinguish between the main assets ELG is catering for, we frequently use the term “Language Resource(s)” (or “data resource(s)” or “dataset(s)”) to denote language data, while we use the term “Language Technology tool/service” to denote software tools performing a a language processing or Language Technology-related operation.
Language Technology tool/service	A tool/service/any piece of software that performs language processing and/or any Language Technology related operation
Non-functional content	Any non-executable type of resource (data resource or source code of tools/services) that can be included at the European Language Grid

Abstract

This document provides an overview of the architectural design of the ELG platform with a particular focus on the ELG platform backend, while the complementary deliverable D3.1 focuses on the ELG platform frontend. We outline the ELG platform's main building blocks and components, the technological considerations and decisions made, the software and application frameworks chosen, the application programming interfaces designed, as well as the operations and user interactions currently foreseen. As the ELG platform development is a dynamic and iterative process, this deliverable is to be considered a living document (until M30), and as such it will be updated to reflect the evolution of the platform design and development. The document provides a functional view of the ELG platform and describes the current ELG platform architecture as well as the development roadmap as it stands at the time of writing. It provides a detailed account of the ELG backend components with a special focus on the ELG platform catalogue and a brief description of the underlying metadata schema, the ELG platform LT processing services, as well as the ELG platform management and support services.

1 Introduction

We aspire to create and establish the European Language Grid (ELG) as the primary platform for Language Technology (LT) in Europe in a concerted effort to address fragmentation in European LT business and research and to strengthen European LT business. The project designs, develops and will deploy and populate the ELG platform *with* and *for* commercial and non-commercial Language Technologies alike, including both functional (running services and tools) and non-functional (corpora, lexica, terminologies, models, etc.) resources. Using the ELG to identify, find, obtain and integrate LT tools/services and Language Resources (LRs), as well as to contribute to and make available LT tools/services and LRs through the ELG, will create a multitude of benefits for companies as well as for non-commercial and academic organizations using and/or providing LTs.

This deliverable provides an overview of the current state of the architectural design of the platform, its main building blocks and components, the technological considerations and decisions made, the software and application frameworks chosen, the application programming interfaces designed, as well as the operations and user interactions currently foreseen. The primary focus of the deliverable consists in describing the ELG platform backend components, while the complementary deliverable D3.1 will focus on the ELG platform frontend. As the ELG platform development is a dynamic and iterative process, this deliverable is to be considered a living document (until M30), and as such it will be updated to reflect the evolution of the platform design and development.

This report is organized as follows: Section 2 provides a summary of the most important findings of the concurrently running user requirements elicitation task (linking to deliverable D2.1). Section 3 provides an overview of the ELG platform from the functional point of view, while Section 4 describes the current ELG platform architecture and the development roadmap as it stands at the time of writing. Section 5 provides a summary of the base infrastructure on which the platform will be deployed and the choices made (essentially linking to ELG deliverables D1.1 and D1.2). Section 6 briefly describes the ELG metadata model and Section 7 provides a short overview of the ELG platform backend. Section 8 provides a detailed view of the ELG backend components with a special focus on the ELG platform catalogue (i.e., the ELG catalogue), while Section 9 describes the main architectural choices underlying the integration and deployment of the ELG platform LT

processing services. Section 10 briefly describes the ELG platform management and support services, Section 11 the current user interface considerations and interactions, and Section 12 concludes the document with a short description of the platform gateway.

2 Initial user requirements

The platform architecture, design and implementation are set to accommodate the requirements of all target users as detailed in deliverable D2.1. In this section we briefly outline the main functionalities that the ELG platform intends to cover; during the course of the project, these will be updated in line with the updates of D2.1.

Users of the ELG system in D2.1 are grouped as follows:

- **Content providers**: providers of LT services and data resources that can be used for the development, enhancement or operation of the LT services; they can be academic, public or commercial organizations or individuals
- **Developers and iIntegrators**: users (e.g., SMEs or startups, but also researchers and developers of LT applications) who wish to deploy services or datasets provided through the ELG platform in order to integrate them in their own products and services and/or use it to conduct research and develop applications
- **Information seekers**: mainly individuals who wish to find information on LT-related events, training resources, etc. (i.e., not primarily interested in the LT marketplace)
- **Information providers**: commercial and academic organizations, LT-related/interested associations/networks and individuals that wish to provide information on events, training, etc.
- **Citizens**: individuals that wish to be informed about LT and how it will influence their lives, and understand the scope of ELG in advancing LT
- **ELG platform and content administrators**: ELG technical partners that will administer and monitor the day-to-day operation and performance of the platform and its contents.

Users may play several roles when interacting with the system, but with different needs each time.

In response to the user requirements, the ELG platform will include mechanisms for the following functions:

- **Functions for content providers**:
    - Providing content-type specific specifications, documentation and help for preparing and packaging content
    - Execution of the functional content at the cloud infrastructure if provided as docker images in containers, following ELG technical specifications (cf. Section 9)
    - Registration of content in different modes, depending on its type:
        - Link to container images at a dedicated ELG docker image registry or similar registries, from which ELG will pull the image
        - Link to functional content with access provided in the form of an API
        - Upload of files directly in the platform for datasets and downloadable tools
        - Link to external repository (e.g., GitHub, institutional or other repository, etc.) where the content will remain and be accessed from

    - Support for the formal description (metadata records) of the content and its related entities (creators, funding projects, publications, related documentation, etc.) and for its update via one of the following modes
        - Upload of a formalized description file
        - Manual editing of the description on a GUI form
    - Assistant for the selection of a licence appropriate for the content (wizard)
    - Support of pre-defined billing schemes, if the provider chooses to share the content for a fee, and managing of billing transactions with the consumers
    - Support for providing new and additional versions (e.g., with reduced functionalities for testing or demo purposes) of the content with appropriate linking to the older/original versions
    - Removal of the content and metadata records under safeguards, e.g., as long as future usage of the content has not already been paid for by a consumer
    - Monitoring content usage analytics
    - Extra features for paid services (e.g., promotion of content, switching on/off user-generated content)
- **Functions for content consumers** (i.e., developers and integrators, but also citizens or other users who decide to deploy the content):
    - Browsing of a structured catalogue with metadata descriptions for all content
    - Searching for content with a faceted menu (on selected metadata elements) or a free-text box
    - Display of the search results in a list with a minimal description for each item and of a detailed view page (landing page) with the full metadata record for each item of the content
    - Preview of data from the content files (depending on the format)
    - Testing of the tools/services if the providers have supplied such a version
    - Execution of specific functional content types (e.g., information extraction, machine translation services) and display of results
    - Deployment of the content in different modes depending on its type, e.g., execution of functional content on the ELG cloud infrastructure or remotely, download of data resources or downloadable tools, etc.
    - Support for the selection of one of the pre-defined billing schemes for the content, various payment options and monitoring of the content usage in accordance with the billing scheme
    - Provision of an API to language processing and other related services for integrators and developers
    - Support and helpdesk for all functionalities
- **Functions for information seekers and citizens**:
    - Browsing information related to the ELG project and the platform, LT-related events, vacancies and training material (videos, texts, etc.) structured in a pre-defined way
    - Search of the CMS content with free-text queries and user-selected prioritization of search results
    - Browsing and searching the ELG catalogue of actors, projects, LT applications and LT business areas
- **Functions for information providers**:
    - Addition and update of information related to entities (actors, projects, LT applications, etc.) via an editor

- Addition and update of information related to events, vacancies and training material via the CMS (to be approved by ELG editors/system administrators)

- **Functions for ELG system administrators and content editors**
  - User management: user registration, user authentication, assignment of user roles and user rights on content, etc.
  - Overall management and day-to-day monitoring of the platform performance and system activities with appropriate notification mechanisms
  - Monitoring of usage analytics and billing/payment transactions
  - Monitoring and removal of content when deemed necessary
  - Follow up on remotely offered content (e.g., checking availability of remote services)
  - Operating metadata harvesting processes, bulk uploading and editing of metadata records
  - Editing and approval of CMS content.

3 ELG platform overview

ELG aspires to become the primary platform for LT in Europe, a one stop shop for LT research, development, evaluation and commercial deployment. It brings together under one platform:

- **Functional content:** language processing tools/services of different types (e.g., machine translation, speech recognition, information extraction) and media input/output (e.g., text, audio, video)
- **Non-functional content**: LRs featuring different types (e.g., corpora/collections of blogposts/tweets, recordings of parliamentary debates, lexica and terminologies, models) and different media (e.g., text, audio, video)
- **LT-related meta-information**: information about language technologies and their descriptions, research and commercial organizations providing LT solutions, LT initiatives and projects, business applications, research and commercial players in different application areas and countries.

ELG caters for and brings together the needs of the following main classes of users, as identified during the initial user requirements study:

- LT-related information providers, seekers and citizens, that is research and commercial organizations and individuals looking for language processing tools/services, language resources as well as LT-related meta-information,
- language processing tools/services and LRs providers (also referred to as content providers in Section 2), as well as LT-related meta-information providers who wish to make available and deploy services and data through ELG, e.g., LT companies, research organizations, universities and PhD students, etc.,
- users/consumers and integrators of language processing services and LRs (also referred to as content consumers in Section 2), e.g., commercial and research technology players, researchers from disciplines other than LT (such as bioinformatics, social sciences, etc.) who need tools and services for different types of text processing (information extraction, text annotation, etc.), audio/video processing (speech recognition, speaker identification, etc.), multilingual data processing (machine translation, computer-aided translation, etc.).

ELG will offer its users multiple alternatives of using the ELG platform and interacting with it, facilitating adaptation to their user scenario requirements, constantly collecting and integrating additional language

processing services and resources, as well as offering technical and linguistic support and gathering user feedback.

3.1 ELG platform metadata

All types of LT assets mentioned above, i.e., software tools/services and LRs and information describing them, as well as all LT-related meta-information are brought together, aligned and interconnected. The collected information is formally structured and harmonized in ELG using a metadata model catering for the full language data and services lifecycle, i.e., integrating, in an appropriately structured way, information about, inter alia, the tools/services and LRs ownership, type, format and technical features, tool/service input/output specifications, service deployment requirements, terms of use and licensing – for a detailed description of the model, see also Section 6.

On the basis of the ELG model, metadata appropriately describing LT tools and services, LRs as well as all LT related meta-information will be indexed and inventorized to create the ELG platform catalogue.

The ELG catalogue provides the user with all mechanisms and functionalities required to search LT tools/services and LRs, and access them depending on and fully respecting their terms of use. Search functions include simple keyword search as well as faceted search on the basis of selected elements of the metadata model. Based on ELG metadata, functionalities of the platform also provide the user with indications on which language tools/services can potentially be applied on which language data.

The ELG platform offers to providers of LT tools/services and LRs all mechanisms and functionalities to appropriately describe, identify (by using/aggregating existing persistent identifiers), ingest and store their assets. Multiple channels are foreseen for their provision and storage, including metadata editors, batch ELG compliant metadata import and harvesting on the basis of agreed harvesting protocols.

3.2 ELG LT tools and services

Of primary importance and one of the main novelties that ELG is introducing, is the ability for LT tool/service providers to describe, integrate, and deploy their functional services through ELG. ELG adopts the well tested approach of containerization; i.e., an OS-level feature that allows multiple isolated user-space instances of the corresponding service. For containerization Docker is (by far) the most popular choice today; it allows to package an application with everything that it is required into one (standalone/self-contained) image.

In LT, containerization has only recently started being adopted, as, for example, in the LAPPS GRID project [Ide et al. 2016]. For LT, containers are considered to be a solution for tackling some of the technical interoperability problems. Operating system, programming languages, frameworks and library dependencies, as well as dependencies of language processing services on different models can all be included in a single image, ensuring that the service will operate as foreseen by its developer. Thus, basic language processing services (e.g., tokenization, sentence splitting) as well as complex language services (e.g., sentiment analysis, event extraction, speech recognition) are encapsulated in standalone/self-contained images. Syntactic and semantic interoperability is tackled inside a single image making sure that e.g., all components of a linguistic annotation or information extraction pipeline interoperate with each other and are correctly chained together. Furthermore, containerization for LT research and development provides (a) support in facilitating reproducible and replicable LT research and development, in the sense that processing software can be deposited and reused as is, (b) further promotion of the principles of Open Science, and (c) assurance that legacy tools,

services and applications depending on older versions of operating systems, programming languages, etc. are still runnable and usable.

ELG will provide a set of guidelines and technical specifications for building images out of existing and emerging language processing software. Containerized processing tools/services (images) coming out of ELG partners, the winning pilot projects of WP6, as well as other interested academic and commercial LT providers will be uploaded and made available through the ELG platform.

The ELG base infrastructure will be able to receive, manage and run many images in parallel. This is considered to be an important service for those organizations that cannot or do not want to host the tools/services themselves (because, for example, they don't have the necessary infrastructure). Therefore, the ELG base infrastructure provides (a) Docker registries for all uploaded images increasing ease of use and speed of deployment, and (b) a flexible and easily scalable environment for controlling and executing the containerized LT tools/services based on Kubernetes[1] (see D1.2 and Section 5 of this document).

Catering for the cases where LT providers may not wish to provide their tools/services in a container, or where the tools/services cannot be easily containerized, ELG, offering a sort of a brokering service, integrates them as remotely invoked services. Such remotely invoked services must comply with RESTful APIs that ELG will define. Service Level Agreements with providers of such remotely invoked services will be put in place together with technical mechanisms for monitoring services availability.

3.3 ELG Language Resources

The catalogue of the ELG platform will be populated with LRs (non-functional content) and associated metadata provided by the ELG partners themselves and through infrastructures and repositories managed by ELG partners. ELG builds upon the META-SHARE policies for resources' sharing and attribution, and it will include openly available and accessible resources, as well as resources that are available with restrictions, including obligations for a fee.

Given the goal of ELG to act as the primary platform for LT in Europe, the ELG catalogue will build bridges to all existing initiatives and reach agreements for harvesting and storing metadata and resources under mutually agreed conditions and attribution, as well as business policies. Thus, ELG will act as a living observatory of LT, consolidating existing and legacy tools, services, LRs, and information as well as newly emerging ones.

3.4 ELG LT-related meta-infomation

In addition to language services and data, ELG will also collect and make available for browsing and searching information about what we call "LT related meta-information", e.g., information on actors (companies, individuals, research organizations, etc.) from the broader LT area, LT business areas, etc.

This meta-information leverages the experience, content and data from the LT-World Portal[2]; the LT-World's data are used for the initial population of the respective part of the ELG catalogue.

Mechanisms and functionalities, including crowdsourcing, web forms, etc., will be put in place for enriching this type of content, in collaboration with the ELG Grid Community.

---

[1] https://kubernetes.io

[2] LT World (http://www.lt-world.org), an ontology-driven web portal aimed at serving the global LT community, was created by DFKI [Jörg et al. 2010]. Although the portal is no longer in production, we plan to adapt the ontology and portal contents to the ELG objectives.

3.5 ELG services

Table 1 presents in a concise way the different types of ELG offerings.

Type	Description	ELG will initially be populated with
Services and tools	Users of the ELG will be enabled to register, describe, upload, search and deploy as well as integrate containerized services, from simple tokenization or part-of-speech tagging to complex processing workflows.	GATE, GATECloud, UDPipe, TILDE's and University of Edinburgh's MT services, SAIL LABS ASR, KWS, sentiment, age and gender detection tools, etc.
Remote services and tools	Users of the ELG will be enabled to register, describe, search and integrate functional remote APIs.	Research and commercial services, among others, GATECloud with 65+ services, UDPipe, TILDE's and University of Edinburgh's MT services, SAIL LABS and EXPERT SYSTEM services
Language resources, services, tools, software code	Users will be enabled to upload, describe, search, download datasets, source code packages, trained models, embeddings, etc.	Datasets, models, tools, lexica, etc. from META-SHARE, ELRC-SHARE, and ELRA catalogue, ELRA's services for e-licensing and e-Commerce
Information about services, tools, resources	Non-functional catalogue entries, for example, on a Language Technology provider company or research centre.	Information collected through projects and initiatives such as, among others, CRACKER, ELRA, ELRC, META-SHARE, LT World, etc.

Table 1: ELG layers of content

ELG is open to all potential stakeholders of the LT and the wider AI communities. For those services that either concern access to LRs with restrictions of use, language processing services that entail restrictive measures (e.g., quotas), as well as for all services that support the integration of new services, datasets or meta-information, a user registration service is set up. Registered users will be authenticated using established protocols and authorized for certain tasks (editing, publishing, commenting, running services, etc.). User management services will also enable the ELG platform to facilitate commercial use of services through possible billing functions like volume/CPU based billing, and experiment, at the level of a prototype with online or offline payments for use of the services.

Catering for the provision of stable service provision of both the ELG platform and the remotely invoked language processing services, the ELG platform will be endowed with a range of monitoring services, including: analytics of the use of the ELG platform, monitoring the availability of the different platform services, and monitoring the availability of remotely offered services.

Access to the ELG platform, the LT tools, services and resources is provided through the feature- and content-rich ELG website (portal). Using modern web development frameworks, the ELG website will include all necessary information, documentation, training materials in different media (textual, video, etc.), facilities for collaboration, user feedback (in the form of comments, reviews and possibly a form of multi-dimensional rating of the functional and non-functional Grid content) as well as blogging and social media.

4 ELG platform architecture

4.1 Overview of the ELG platform architecture

An overview of the ELG platform architecture is depicted in Figure 1.

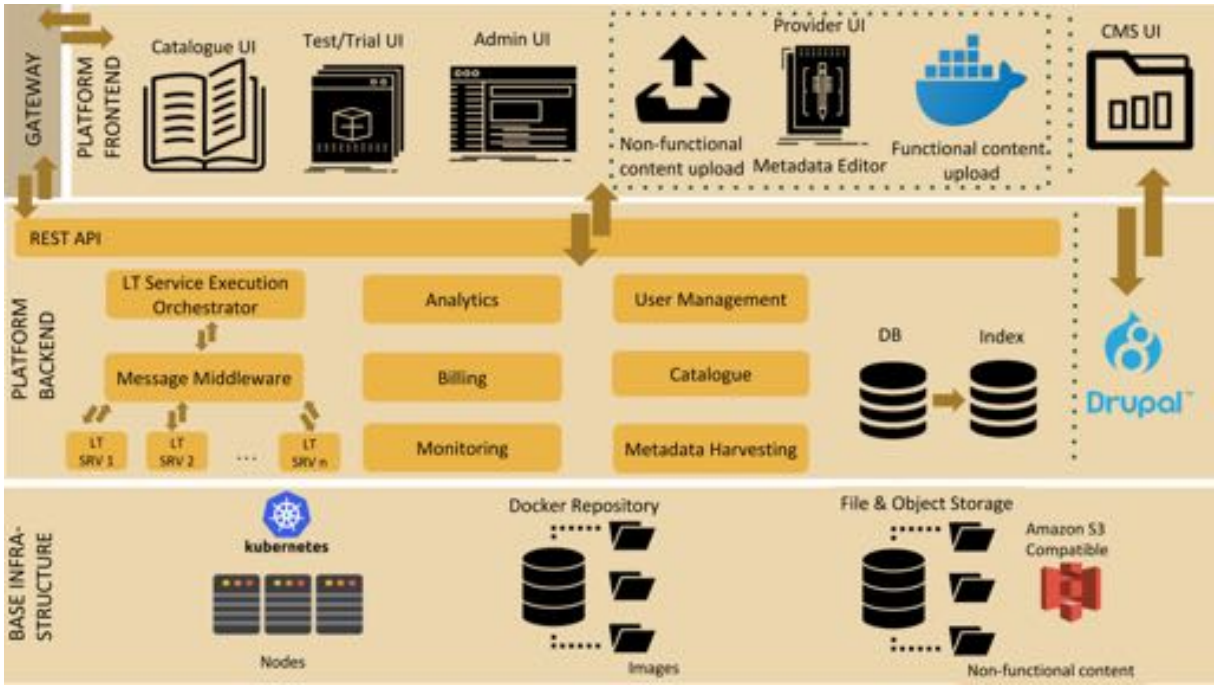

Figure 1: Architecture diagram

The ELG platform is divided into three main layers: the **base infrastructure**, the **platform backend** and the **platform frontend (user interface, UI)**.

The base infrastructure (Section 5) is the layer on which all the software components are deployed and run; it includes the supporting tools that facilitate development and management of the ELG software.

The platform backend layer consists of all the components that empower ELG: for example, core components such as the database, the index, the user management component as well as the LT tools/services; a detailed description of the backend is given in Sections 7 to 10.

The UI layer, briefly presented in Section 11, consists of the web pages of the CMS and the pages that render the information that is stored in the database (metadata), pages for registering resources, the catalogue and search interfaces, etc. The latter UI (non-CMS) uses/consumes REST services exposed by the ELG platform backend (e.g., catalogue application) in order to get the information needed for creating the respective pages.

4.2 Technical development roadmap and Minimum Viable Product (MVP) specification

In line with the ELG Description of Action, the development and roll-out of the ELG platform will be performed in three major platform releases. The diagram in Figure 2 depicts the current development roadmap, essentially annotating the ELG architectural diagram with the development priorities A (highest priority), B (medium priority) and C (lowest priority). These priorities reflect and include the main critical functionalities to be offered by each release.

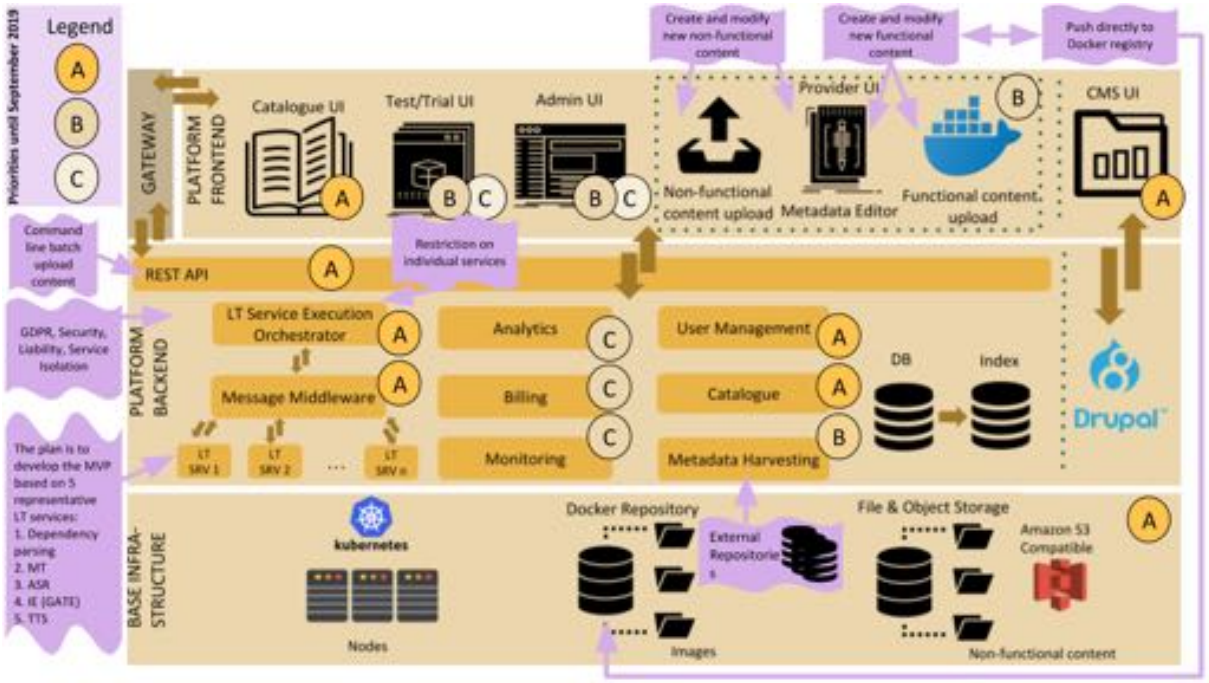

Figure 2: Technical development roadmap and MVP

The features and functionalities as well as the LRs and language processing services targeted by each major release are depicted in Table 2.

Release	Features	Due
D2.4 ELG platform (first release)	• backend components required for the operation of the catalogue: simple user management component, components supporting documentation/uploading, storing/downloading of all resource types (tools and services, datasets, etc.), APIs required for interacting with other layers • first version of the guidelines on its use and provision of resources, instructions for containerization and invoking of remotely accessible web services • limited sets of tools and services and LRs	M16
D2.5 ELG platform (interim release)	• updated version of the platform including the components and APIs required for running language processing services (containerized services stored in the ELG and web services via REST APIs) directly in ELG • updated (as/if needed) version of the guidelines on its use and provision of resources, instructions for containerization and invoking of remotely accessible web services • updated catalogue with resources from ELG partners	M26
D2.6 ELG platform (final release)	• updated version of the platform including management and maintenance of the platform: monitoring of the platform, monitoring of remotely offered services, platform usage analytics, prototype version of user billing and payment services • updated catalogue with resources from ELG pilots and collaborating initiatives	M34

Table 2: Platform releases and features per release

5 Base infrastructure

The base infrastructure is the layer on which the ELG platform (backend and UI) are installed and run along with the tools for managing the ELG-related software development. It (mainly) consists of the following parts: a kubernetes cluster, a docker repository and a file and object storage.

5.1 Kubernetes cluster

The ELG platform software (backend and UI), can be divided in two parts:

- the core components, e.g., the catalogue database, the metadata index, the web server, the catalogue application, etc. which are being developed and maintained exclusively by the ELG technical team, and
- the set of LT tools/services that are integrated in the Grid and are developed either by consortium partners or by external providers.

As discussed in Section 3.2, all aforementioned tools and services will run as containers since this facilitates management and deployment. Currently, the most popular, open source, and robust system for managing, scaling and orchestrating containerized applications is Kubernetes (in short k8s)[3]; for this reason, the ELG consortium has decided to adopt it. Kubernetes works with a range of containerization tools; among them, the Docker solution is by far the most widely spread and is thus selected also for ELG.

As prescribed in the ELG proposal, in the beginning of the project a subcontractor would be selected for WP1, i.e., a cloud provider that will be in charge of installing, running and administering (on a daily basis) the base infrastructure, under the supervision and responsibility of the ELG WP1 leader (the ELG coordinator). The selection process has been completed and the Berlin-based company SysEleven[4] has been chosen; see D1.1 and D1.2 for more information.

After selecting the cloud provider, the ELG consortium collected the (hardware/software) requirements for a proof-of-concept deployment of the ELG platform: i.e, a deployment that includes all core components (e.g., database, catalogue application) as well as a small number of LT processing services. Based on these requirements and some initial discussions a development k8s cluster has been configured and delivered by SysEleven to the ELG consortium.

In order to deploy an application in k8s, a set of config files (in YAML format) that specify the required information must be created. Examples of such information are the Docker image location, the number of replicas that will be deployed, the ports that will be exposed and the name that will be given to the deployed service; the latter can be used by other k8s applications/services for accessing it. In ELG we have to manage several software components and we also need different deployments of the platform, such as a single-node deployment for development/testing, a deployment in the SysEleven cluster, etc. For these reasons we have decided to use Helm[5], a package manager for k8s that automates/facilitates deployments. Helm uses (generic) templates for describing the k8s resources. For each different installation/deployment the templates are filled

[3] https://kubernetes.io
[4] https://www.syseleven.de/
[5] https://helm.sh/

in with the respective configuration data by running a simple script. The generated config files are then submitted to the cluster via the respective k8s API.

5.2 Source code and Docker repository

For hosting and managing the source code of ELG software components an account has been created at GitLab[6], which is a source code repository and a development platform (for Git projects) similar to GitHub[7] and BitBucket[8]. GitLab was chosen because it provides for free numerous features, such as unlimited number of private and public repositories, a built-in Docker registry for storing images and built-in continuous integration (CI) and continuous development (CD) functionalities. In GitLab, CI is automatically triggered when a commit is pushed in a Git repository; a YAML file (.gitlab-ci.yml) defines the CI pipeline that will be executed. By default the pipeline runs at GitLab shared servers; however, it can also run in a dedicated machine by installing the respective software (GitLab Runner[9]). A CI pipeline typically includes source code compilation, running unit tests and building the respective executables. There is also an option to add code for automatically creating the respective Docker image and pushing it to GitLab's built-in Docker registry.

5.3 Storage

For hosting/storing the ELG non-functional content, i.e., corpora (raw or LT processing results), statistical and machine learning models, as well as lexical/conceptual resourses (e.g., lexica, ontologies, term lists) used in LT processing, a storage solution is required. We plan to use an S3-compatible Object Storage solution that is hosted by SysEleven for this functionality.

We are also exploring the option to create a generic ELG-specific Storage API interface that will include methods for all required storage operations in order to make the platform more modular and flexible. One implementation of this interface will use SysEleven's S3-compatible storage (as already discussed) and another will use a local disk; a configuration file will specify which solution will be used. The local disk implementation may prove useful for a local, private deployment of ELG; e.g., an installation running on the premises of an institution/company without internet access for running LT services on sensitive data.

6 ELG metadata model

The **ELG-SHARE metadata model** (or alternatively, in short, ELG metadata model) will be used for the description of all entities of interest to the ELG target users. It constitutes the backbone of the ELG catalogue, which brings together language processing services and tools, LRs (datasets of different types and media, models, lexica, terminologies, etc.) as well as agents, activities, technologies and business application areas related to LT (Figure 3).

---

[6] https://gitlab.com/european-language-grid
[7] https://github.com
[8] https://bitbucket.org/product/
[9] https://docs.gitlab.com/runner/

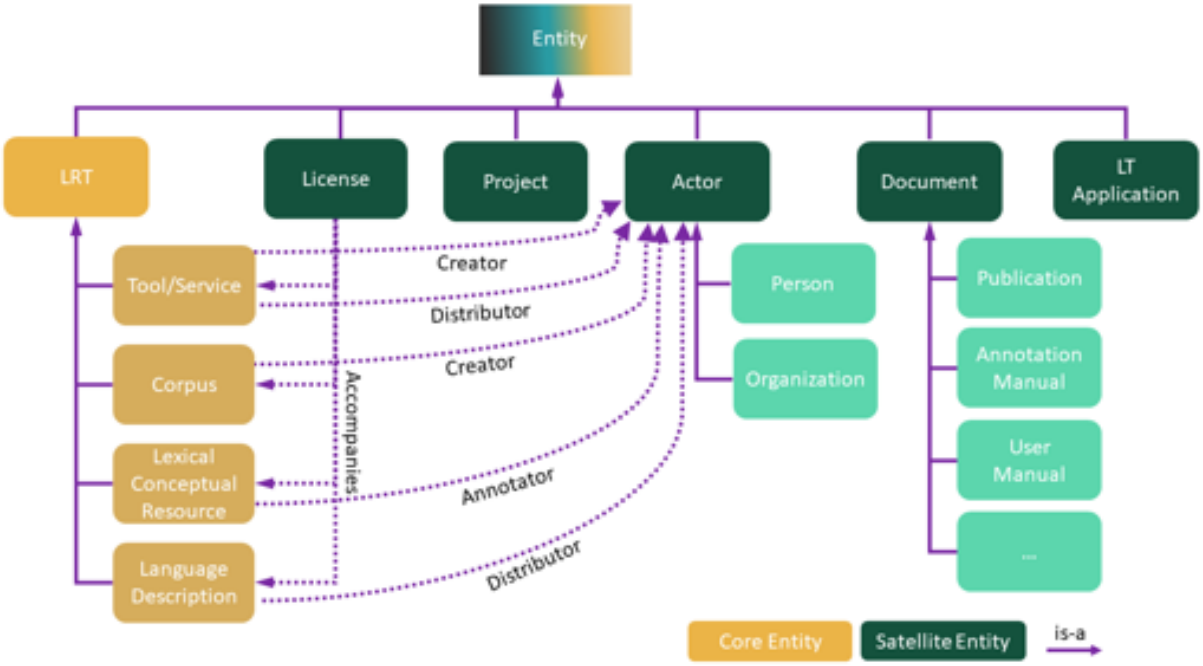

Figure 3: Overview of the ELG-SHARE entities

Its main objectives (to be further refined during the project course in order to accommodate the evolving user requirements, see D2.1) include the following:

- cover needs of discoverability/findability of resources and related entities
- satisfy documentation needs for all of them at different levels of granularity
- address (at the metadata level) interoperability requirements of resources belonging to the same types and media, but coming from different sources with different descriptions, as well as between resources of different types and media (e.g., between datasets and services to be used for their processing)
- facilitate accessibility by human users and, where possible/required, machines (e.g., by stating direct download/execution locations)
- provide an overview of the LT landscape allowing users to navigate through applications, products, datasets, actors, projects, etc. linked through the LT activities they are engaged in
- act as a bridge between the catalogue and the CMS information content where appropriate, e.g., the training material intended to promote awareness of LT to users (citizens, SMEs, industry, etc.) interested in using it but less knowledgeable about it.

The main principles and strategies employed in the design of the ELG-SHARE schema consist of the following:

- adopt and adapt the FAIR principles[10] [Wilkinson et al. 2016] to the needs of ELG
- re-use or map to existing standards and best practices, especially other metadata schemas and vocabularies, and previous relevant initiatives for the documentation of LRs and LT tools/services, etc.
- be flexible enough to support varying degrees of documentation completeness
- organize the schema elements and accommodate common vs. particular features of resource/media types

[10] https://www.force11.org/group/fairgroup/fairprinciples and https://www.go-fair.org/fair-principles/

- normalize user input where appropriate but also allow for free user input.

The model builds upon previous work from (a) the META-SHARE metadata model[11] [Gavrilidou et al. 2012], which caters for the description of language resources and language-processing technologies, its application profiles, namely ELRC-SHARE[12] [Piperidis et al. 2018], OMTD-SHARE[13] [Labropoulou et al. 2018], and CEF-AT, which extend, restrict and adapt the basic model to specific domains and areas (e.g., public domain resources, text and data mining domain, etc.), and the MS-OWL[14] [McCrae et al. 2015], which is the RDF/OWL representation of the model; (b) the LT-World and other catalogues and projects for the enrichment of the descriptive model of related entities (Figure 4). More recent developments in the metadata area at large are also taken into consideration for the consolidation of the new model. The proposed model is currently under discussion and revision by members of the Consortium. A pre-release will be made available for review and feedback to a wider group with experts from the Grid community, mainly NCCs and the ICT-29b projects, as well as to other potential users, through face-to-face dissemination events and electronic communication. The final model will be documented in D2.3 (due M8); minor adjustments may be needed during the project lifecycle, which will be documented together with the platform releases.

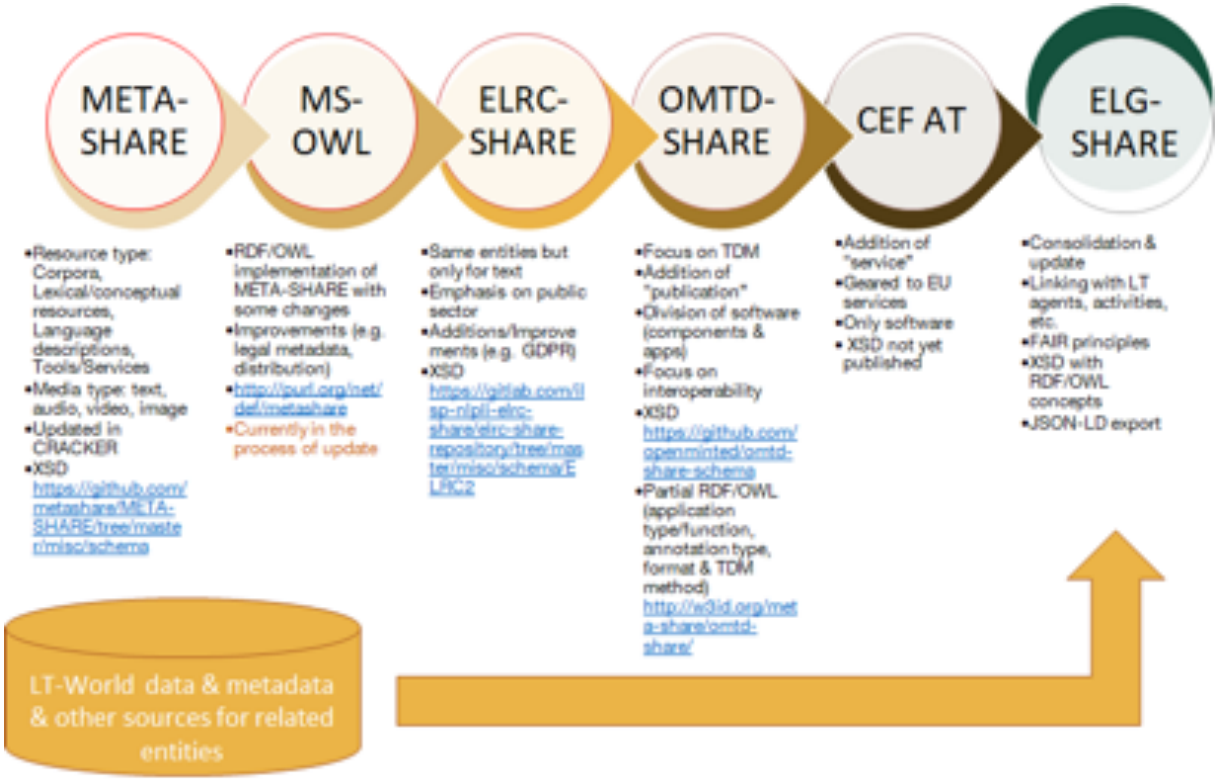

Figure 4: Evolution of the ELG-SHARE model

The ELG-SHARE data model comprises of the following entities:

- **language resources and technologies**, further classified into:

[11] http://metashare.ilsp.gr/knowledgebase/homePage
[12] https://gitlab.com/ilsp-nlpli-elrc-share/elrc-share-repository/tree/master/misc/schema/ELRC2
[13] https://github.com/openminted/omtd-share-schema and http://w3id.org/meta-share/omtd-share/ (RDF/OWL implementation of part of the schema)
[14] http://purl.org/net/def/metashare

    - **corpora**, i.e., datasets of mono/bi/multilingual text documents, audio/video recordings, multimedia datasets, parallel corpora, translation memories, etc.
    - **lexical/conceptual resources**, including lexica, ontologies, term lists, gazetteers, computational dictionaries, etc.,
    - **language descriptions**, which mainly refer to computational grammars, statistical and machine learning models,
    - **tools/services**, i.e., pieces of software offered as locally executable codes or as web services, hosted and running in the ELG cloud or remotely running
- **related/satellite entities**, such as **actors**, be it **persons** or **organizations** that have created or curate the resources, or **projects** that have funded them or where they have been used.

To fulfill the ELG objectives, the metadata model, as regards data resources and processing tools/services, will cater for their full life-cycle, integrating information about the resource identification, type and technical features, deployment requirements, legal rights and obligations, associated documentation, etc. As regards related entities (i.e., actors, projects, etc.), it will include all features required for their identification and all necessary information describing and promoting relevant activities and products, taking into account, where appropriate, GDPR issues for persons.

All this information leads to a complex and demanding schema; to ensure flexibility and uptake by resource providers, the elements are classified into three levels of optionality:

- **mandatory:** elements that are necessary for intended purposes
- **recommended**: elements that can help the current or future use of the resource, or useful information that providers have not yet standardized
- **optional**: all remaining information.

In addition, all the properties and relationships are organized in a modular structure, following the principles of the component-based mechanism (Component MetaData Infrastructure, CMDI), according to which semantically coherent elements are grouped together to form components [Broeder et al., 2008], see Figure 5. For instance, the licensing component includes elements such as the name and URL of a licence, attribution text, copyright holders, information on fee, etc. Some of these components, mainly those including administrative features, are common to all resource types, while the technical features are particular to the specific resource and media types.

The schema will be made publicly available in the form of an XML Schema Definition (XSD) including elements from an RDF/OWL ontology, thus adhering to FAIR principles and facilitating the linking to other schemas and catalogues.

At this stage, we have specified only a subset of the mandatory and recommended elements pertaining to tools/services and text corpora that will be used for the MVP. This is used for the design of the database and a draft documentation of the entries that will populate the initial catalogue.

Figure 5: Simplified subset of the metadata schema

7 ELG platform backend

The ELG platform backend comprises three major components: the ELG platform catalogue (in short ELG catalogue), the ELG (platform) language processing backend services and the ELG (platform) management and support backend services. They are described in detail in Sections 8, 9, and 10 respectively.

8 ELG catalogue backend

The ELG catalogue backend is designed and implemented so as to provide all the functionalities needed for the unobstructed and smooth operation of the platform. The functionalities of the catalogue backend, to be served through a secure REST API, include:

- Metadata management, i.e., creation, retrieval, update and deletion of metadata records
- Content management, i.e., upload of content (functional and non-functional) associated with metadata records
- Metadata indexing for search and fast retrieval
- User management, i.e., user authentication and authorization through well defined access control policies
- LT services requests, i.e., dispatching and handling requests for LT processing services that are hosted by ELG
- Billing information

The ELG catalogue backend mainly consists of three components:

- The core software that implements the REST API (Django >= 2.2.2)

- The database software used for persistent storage of the data needed for all the platform operations (PostgreSQL >= 11.3)
- A search and analytics engine aiming to index a subset of the data stored in the database, in order to provide full text and faceted search functionality on the metadata, as well as usage analytics (Elasticsearch >= 6.8.2)

8.1 ELG catalogue backend development framework

The needs of the ELG catalogue backend determine the criteria for selecting the most suitable web development framework. These needs include a rapid development environment for building RESTful APIs, without, however, compromising reliability, scalability and security. Furthermore, community support is a strong indication of the popularity and foundation of a given framework.

Django is an open-source framework for web applications based on Python. Oriented towards rapid development and pragmatic, clean design, it features simplicity, flexibility, reliability, and scalability. It is also supported by detailed documentation provided by the Django Software Foundation (DSF)[15] and comes with many out of the box features that favour rapid development, while at the same time it is fully customizable in order to suit the needs of a specific application. A large collection of over 4000 useful packages and utilities are also available, which demonstrates the popularity of the framework and the size of the Django community, in terms of contributions and support. It is used by a considerable number of companies and organizations like, Bitbucket, Instagram, NASA, YouTube, DropBox, etc., and has also served as a reliable solution for the development of repositories led by ELG partners, such as META-SHARE[16], ELRA catalogue[17], ELRC-SHARE[18], CLARIN:EL[19], QTLaunchPad[20] and the catalogue of CEF eTranslation services[21].

In terms of scalability, since the components of the Django framework are not tightly coupled, they can be unplugged and replaced whenever the application requires more specific solutions. Security is handled by hiding the source code from direct viewing on the Internet, along with built-in protection against cross-site scripting attacks and SQL injection attacks.

Django supports almost all databases, and specifically provides ready to use and strong solutions for relational databases. The Object-Relational Mapper (ORM) is one of the key features of Django, providing an elegant and powerful way to interact with the database in a seemingly unified way, hiding the details required by each database.

8.2 ELG database and indexing

Database solutions come in two main types: SQL (RDBMS) and NoSQL – in other terms, relational databases and non-relational databases. The key differences between the two are the way they are built, the type of information they store, and the way they store it. While SQL databases (e.g., MySQL, PostgreSQL, etc.) structure the information in tables that connect to each other with relations, NoSQL databases (e.g., MongoDB,

[15] https://www.djangoproject.com/foundation
[16] http://www.meta-share.eu
[17] http://catalog.elra.info/en-us/
[18] https://elrc-share.eu and redeployments of it like ELRI (http://www.elri-project.eu/)
[19] https://www.clarin.gr
[20] http://qt21.metashare.ilsp.gr/
[21] https://cef-at-service-catalogue.eu/

Cassandra, etc.) are document-oriented, that is, they store the information in a single "Document" object within "Collections", much like file folders.

The main reason for choosing NoSQL over SQL is usually performance and the need for JSON storage. Performance in NoSQL is considered better when compared to that of an SQL database; however, write safeguards and database transactions are not inherently supported by NoSQL. NoSQL solutions are, therefore, considered to be prone to mainly integrity errors and increase the risk of corrupted data. Furthermore, current security issues, such as lack of encryption support for the data files, vulnerability injection attacks (via JavaScript or string concatenation), denial of service (DoS) attacks, make NoSQL a weak choice [Okman, Lior et al. 2011]. JSON storage can be easily applied in an SQL database, especially PostgreSQL, which provides a searchable JSON field.

For the above reasons, the ELG platform backend adopts a relational database solution for the ELG catalogue (i.e., LRTs and satellite entities metadata), billing information and other administrative data as well, such as user management, storage and data processing information and overall usage statistics. Selected data from the database will be indexed by a search engine that will handle user queries on the metadata and provide analytics on statistical information.

8.2.1 Database requirements and criteria for selection

Sensitive information, like user management, billing and administration, require security and reliability on database transactions. PostgreSQL and MySQL are among the most popular relational databases. Although MySQL appears to be the most commonly used[22], PostgreSQL is more capable in terms of schema support for Django, whereas MySQL lacks support for transactions around schema alteration operations. Schema alteration operations is a built-in feature in Django (Django Migrations) and is considered as highly crucial during the rapid and continuous development phase and beyond. Moreover, the Django ORM provides support for PostgreSQL specific fields (JSON, Array), which are well suited for the metadata model implementation.

PostgreSQL is committed to conform to the ANSI-SQL:2008[23] standard, in order to support the official features of the SQL language, thus facilitating portability of the SQL code across database systems. It is fully ACID (Atomicity, Consistency, Isolation and Durability) compliant, and is well-known for its solid referential and transactional integrity. Validated data storage is ensured by primary keys, restricting and cascading foreign keys, unique constraints, not null constraints, check constraints and other data integrity features.

In terms of data replication for improved availability, the ELG backend will use PostgreSQL' s synchronous replication feature between the master and the slave databases. Along with providing user access control, host-based access control, and user authentication, PostgreSQL also natively provides the capability to encrypt client/server communications using SSL as an additional security feature.

8.2.2 Indexing requirements and criteria for selection

Data search and retrieval is one of the ELG backend's main components. In order to provide advanced search capabilities and fast retrieval, it is necessary to utilize an indexing and search engine solution on top of the

[22] https://insights.stackoverflow.com/survey/2018#technology-_-databases
[23] https://www.iso.org/standard/45498.html

primary database. The most popular open source search engines are Solr[24] and Elasticsearch[25]. Both solutions are well established and documented, with large community support and high performance ranking.

In terms of API implementation, Solr implements a Java API with SolrJ – or SolrNet (for Microsoft technologies) – but also provides a set of REST APIs. However, Elasticsearch has a more Web 2.0 REST API, which makes it the preferred choice for modern RESTful applications. The REST API feature of Elasticsearch is suitable for the ELG catalogue backend, which will also expose its services through REST endpoints and is, therefore, the preferred software to be used for the needs of the ELG platform. Moreover, The REST API solution in combination with Elasticsearch has already been adopted by the backend developing team in the implementation of the catalogue of CEF eTranslation services.

Elasticsearch also provides real-time log analysis and visualization and, combined with other tools like Logstash and Kibana, it can deliver high level visualizations of statistical data stored in the database.

Security in Elasticsearch is easily configurable at various levels since version 6.8, which comes with TLS encryption, native and file-based authentication, and role-based access control in the default distribution.

8.3 Lifecycle of data and metadata in the ELG platform

The ELG database will store metadata records, along with other administrative information, such as the existence of a dataset for a given record and the visibility status of a metadata record. The ELG platform backend will distinguish 3 visibility statuses for metadata records:

- **internal**, used as the initial status for all metadata records
- **ingested**, used as an intermediate step before making the record publicly available; accessible by all users of an editor group (see Section 10.1)
- **published**, for finalized metadata records; published records are available for searching and browsing on the public catalogue.

The combination of users' access rights and the visibility status of a metadata record yields enhanced authorization possibilities to the user management module giving metadata record owners extra leverage and control over the visibility and accessibility of the metadata record repository.

Metadata in the ELG catalogue are subject to all CRUD (Create, Read, Update, Delete) operations, based on user authentication and authorization to perform certain actions. The creation of a metadata record in ELG can be carried out either from within a dedicated metadata editor or by uploading files containing structured metadata descriptions that conform to the ELG-SHARE metadata schema. The ELG platform backend makes use of the concept of ownership (creator) of a metadata record by a user and/or user group in order to filter the subset of the records that an authenticated user is allowed to perform actions on. Those actions include:

- Metadata retrieval
- Metadata update
- Metadata deletion

Metadata update is allowed only on the metadata information that are not shared by, and thus affect, other records of the database (e.g., licence information, descriptions of entities such as organizations or persons); the

[24] https://lucene.apache.org/solr/
[25] https://www.elastic.co/

shared information is restricted to users with specific permissions, based on the adopted user management policies. User management policies also determine permissions for full metadata retrieval and record deletion. Resources on the catalogue are assigned unique identifiers, associated with the appropriate metadata records and bound to the same operation policies.

However, while a resource included in the ELG catalogue must be related to a metadata record, the same does not hold in the opposite direction of the relationship, i.e., it is not mandatory for a metadata record to be related to an actual language resource hosted in the ELG platform, since it may describe and just provide a pointer to a resource that is stored elsewhere.

8.3.1 Catalogue population

8.3.1.1 Metadata editor

The ELG catalogue backend is connected to a dedicated UI frontend, which provides an interactive environment to be used for metadata editing. The editor environment exposes the metadata model to the authenticated and authorized users, ready to accept input through a number of user friendly forms, designed by a UX expert.

The editor module caters for the guided creation and update of metadata records with descriptions and recommendations for all input fields and guarantees the validity of the data to be inserted into the database. Input data validation will also be accompanied by well defined interactive messages informing the user about potential input errors.

8.3.1.2 Batch metadata uploading

In addition to the metadata editor, users will be able to create or update existing resources by uploading files that contain metadata descriptions conformant to the ELG-SHARE schema in JSON or XML format. The backend checks, at import time, that the input data are valid and accepts or rejects the user's request, by providing appropriate messages. Batch upload will be available for new metadata records, accepting a compressed archive (e.g., .zip, .tar, etc.) containing the input, while batch update capabilities will be further explored. Basic CRUD and management operations for metadata records are also planned to be available through ELG API clients that will enable the interaction with the platform through a command line interface.

8.3.1.3 Metadata harvesting

Alternative ways of populating the ELG catalogue database include metadata harvesting (potentially with data) from other repositories. The harvesting policies as well as candidate source repositories (e.g., ELRC-SHARE, META-SHARE, catalogue of CEF eTranslation services, CLARIN, etc.) will be explored and well defined source-target schema conversion tools are foreseen according to the preferred harvesting protocol (e.g., OAI-PMH, ResourceSync or proprietary).

8.3.2 Uploading and storing non-functional content

Uploading and storing non-functional content to ELG platform is controlled by the user management module and guarantees that the user has the required permissions to upload a dataset for a given metadata record and also assumes the user's legal liability for the uploaded data. Datasets will be stored either at SysEleven's S3-compatible storage solution or at the ELG-specific storage solution, based on the preferred configuration.

Administrative information regarding the uploaded data such as checksum, identifier, creation date, modification date, related resource, etc. will be stored in a dedicated database table, thus separating the

metadata description from the information needed for the management of the resource itself, which is used only internally by the platform.

8.4 Search, browse, view and download functionalities

A range of database fields (metadata elements), such as resource name, description, languages, licences, format, character encoding, etc., will be selected (and updated according to increasing requirements) for indexing by the Elasticsearch engine in order to facilitate full-text and faceted search primarily on metadata records that are marked as "published" (public visibility status). Faceted search will provide a mechanism to filter search results by combining indexed metadata fields presented to the user as clickable facets that will also bear aggregations on the available facet values. At a preliminary stage the following facets are considered: resource name, resource type, media type, language(s), licence(s). Facets (like indexing elements) are also prone to change responding to emerging user requirements.

The search results feature pagination for rapid rendering of information and browsing by the frontend, while the detailed metadata view utilizes direct calls to the respective endpoints in order to retrieve the full description of the resource.

Downloading of resources will be accessible through the record' s detailed view in accordance with the legal and commercial terms for its distribution, provided that the record has an associated downloadable resource (dataset or tool). During a download request, the backend ensures that the user agrees with the resource' s legal terms and the payment/billing policy if needed, before providing the actual resource. In other cases, the user may be prompted to contact an entity (person or organization) in order to ask for further information on how to obtain the requested resource, if the resource' s licence does not permit direct download.

8.5 Internal APIs between core components

The ELG catalogue backend is fully RESTful, that is, all supported functionalities will be available through the catalogue' s secure REST API, which includes the following endpoints used by the frontend:

- Create a metadata record
- Retrieve a metadata record
- Update a metadata record
- Delete a metadata record
- Retrieve a list of metadata records; based on user permissions, this list may be a subset of the overall record list
- Change the ownership of a metadata record
- Change the visibility status of a metadata record (e.g., ingested → published)
- Upload/Replace/Delete a resource for a given metadata record
- Search for resources
- Request processing of a resource hosted in the ELG platform
- Handle user registration requests
- Handle user authentication and authorization
- Handle billing management

The catalogue API will be updatable in order to accommodate increasing functionality needs and will also expose an automated documentation that reports on all the available endpoints and allowed operations (HTTP

methods). An example of the JSON response of the ELG catalogue API as well as documentation generated by the Core API can be seen in Figure 6 and Figure 7, respectively.

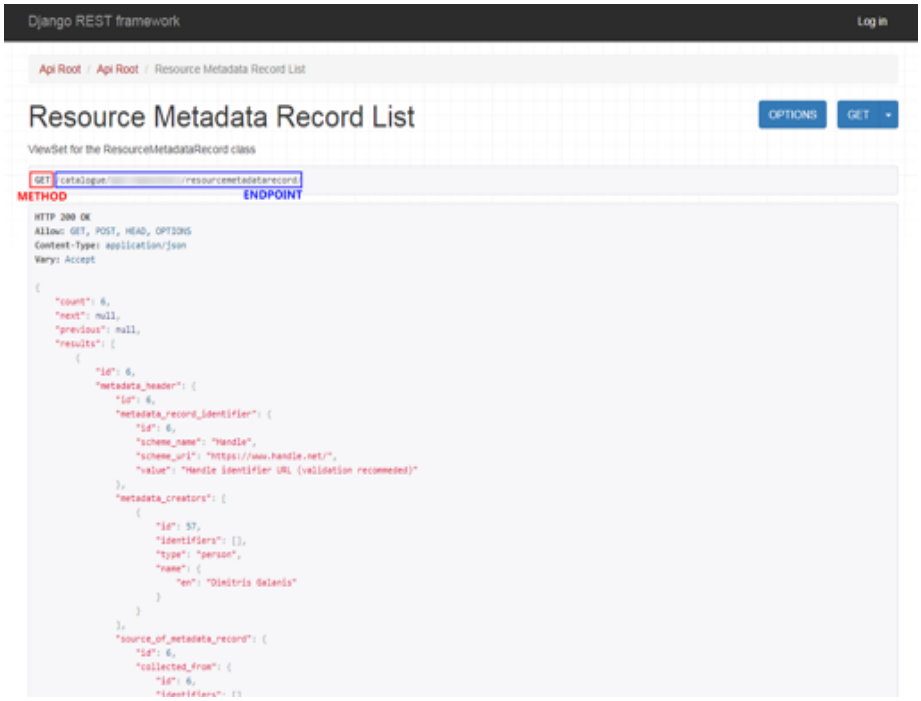

Figure 6: JSON response of an ELG catalogue API endpoint using the GET method

Access to the backend REST API is mainly available through the frontend where all operations are performed from, using JSON Web Tokens for user authentication and authorization and Cross-Origin Resource Sharing (CORS) rules in order to determine the origin of the API calls. Alternative ways of access to the API include the implementation of a command line interface (API client) that will serve the main metadata and data management functionalities.

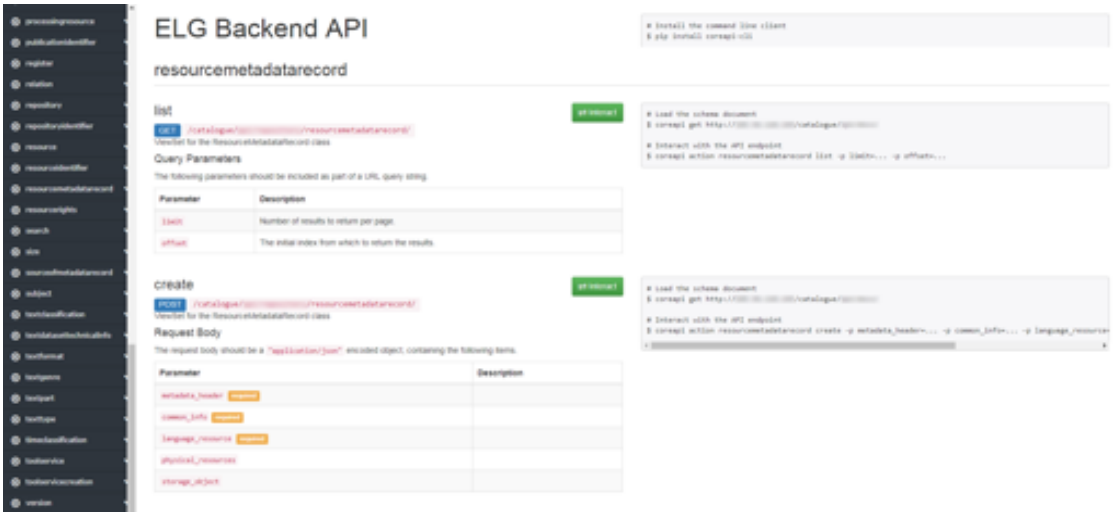

Figure 7: ELG backend API Documentation generated with Core API

The implementation of the catalogue backend API is accompanied by a descriptive documentation for all available endpoints, using the Core API[26] Python package, which will be automatically updated as the enrichment of the API progresses, based on emerging requirements.

9 ELG platform LT processing services

As already mentioned, one of the main goals of ELG is to bring to the same platform a substantial number of LT tools/services of various classes (IE, MT, ASR, etc.). For each such class a specific API is specified (Task 2.5) in order to standardize how the LT tools/services will communicate and be integrated with the rest of the ELG platform; these APIs along with containerization technology (Docker) will facilitate at large extent the integration and the deployment of the LT services. LT service execution will be provided to end-users via an appropriate RESTFul API.

9.1 Messaging middleware

Each containerized LT tool/service communicates with the rest of the ELG platform (i.e., consumes processing requests, produces responses, optionally produces progress reports) via an appropriate messaging middleware and the respective queues. For messaging middleware we have selected the widely used RabbitMQ broker[27] since it provides a variety of configuration options, it is simple and there are also RabbitMQ clients/libraries in various programming languages (e.g., Java and Python). An alternative message middleware solution that was explored and that we might adopt in the future is the Kafka system[28]. Kafka has been reported to achieve (if appropriately configured) high throughput and high availability[29]; these two features might be important, for example, in the case of data processing overload.

9.2 Internal and external (LT) processing APIs

The messaging system and the request/response queues could have been publicly exposed to the internet; in such a scenario the respective clients would be able to communicate directly to the backend LT tools/services. However, we have instead put an LT service execution orchestrator component in the front for the following reasons:

- All functionalities relevant to LT processing are implemented there and not in the catalogue application which is designed for offering services on the metadata of the resources; e.g., search, view, insert, delete, update; see Section 8.5 for more details. In this way we achieve a clean separation between functionalities and the platform (which is more modular).
- For handling appropriately processing requests when the system is overloaded; e.g., put the request in a waiting queue, start an LT service (in k8s) that is not currently available, etc.
- Make easy the integration of the execution backend with the catalogue application and the rest of the platform (e.g., storage). This integration enables, for example, the processing of the datasets that are hosted in the ELG platform and listed in the catalogue; in this scenario the LT service execution orchestrator fetches the data from the storage (e.g., S3), generates the respective LT internal API messages (T2.5), collects the responses and creates the resulting dataset.

[26] https://www.coreapi.org/
[27] https://www.rabbitmq.com/
[28] https://kafka.apache.org/
[29] https://itnext.io/kafka-vs-rabbitmq-f5abc02e3912

The LT service execution orchestrator exposes a REST API which can be used by any client; e.g., the ELG catalogue UI or a command line tool (e.g., curl). In this way we have a common gateway for all types of processing requests (datasets vs. single documents, hosted vs. non-hosted datasets, etc.)

The LT service execution orchestrator is implemented in Java using the Spring Boot framework[30]. Spring Boot was selected since it (a) facilitates application configuration, (b) allows the creation of a consolidated standalone application which can be easily Dockerized and deployed, (c) provides an easy way to create REST services, and (d) provides easy integration with a large number of tools and libraries (e.g., databases, messaging middleware, storage solutions).

Currently, the proof-of-concept (POC) LT service execution orchestrator that we have developed offers just one REST POST service for processing one document; this service is specified below.

Service endpoint	Request	Response
https://{LTServExecOrchIP}/process Document/{ltService}	{ "id": "an id", "content": "data to be processed" }	A JSON with results

LTServExecOrchIP is the IP where the LT service execution orchestrator is running at, and *ltService* is the ID of the LT service that we want to use. The POST payload of the request is a document processing request and the response is the processing result; both request and response are encoded in JSON format. When the orchestrator receives a document processing request it creates an appropriate T2.5 message, pushes it to the respective queue, collects the processing response and returns the result.

The current POC orchestrator has been integrated (via RabbitMQ queues) and tested with two backend information extraction (IE) services that were developed by ELG partners: ANNIE[31] from the University of Sheffield and Cogito[32] from Expert Systems. Each one is an example of the two ways that can currently be used to integrate an LT service in ELG.

- **Direct integration:** One Dockerized application that contains the LT service and directly interacts with the respective RabbitMQ queues; ANNIE is created in this way.
- **Integration via an adapter:** It consists of two Dockerized applications; one adapter that interacts with the queues (as in direct integration) and translates the messages to the actual LT service and the LT service itself. Cogito has been developed/deployed in this way.

An example of interaction with the described “processDocument” web service using the orchestrator instance that is deployed at SysEleven’s k8s development cluster (see Section 5) is the following.

Service endpoint	Request	Response
https://185.56.128.145/dev/rest/processDocument/srv-annie-ie	{"id":"123","content":"lan lives in Sheffield"}	{"metadata":{"id":"123"},"response":{"type":"annotations","annotations":{"Person":[{"start":0,"end":3,"features":{"gender":"male","kind":"firstName","rule":"GazPer

[30] https://spring.io/projects/spring-boot
[31] https://gate.ac.uk/annie.html
[32] https://www.expertsystem.com/products/cogit-cognitive-technology/

sonFirst","firstName":"Ian","ruleFinal":"PersonFinal"}}],
"Location":[{"start":13,"end":22,"features":{"kind":"loc
Name","rule":"InLoc1","locType":"city","ruleFinal":"Loc
Final"}}],"Organization":[],"Date":[]}}}

The ANNIE service is called by using the respective id "srv-annie-ie" and a document is sent for processing ("Ian lives in Sheffield"). The response is in JSON format (actually, it is the LT-internal API message that was returned) and contains the named entities that ANNIE has extracted.

The current version of the orchestrator uses the relevant Spring Boot Swagger[33] libraries to automatically generate documentation (HTML pages) for each exposed web service. This way can keep more easily synced the code with the respective documentation.

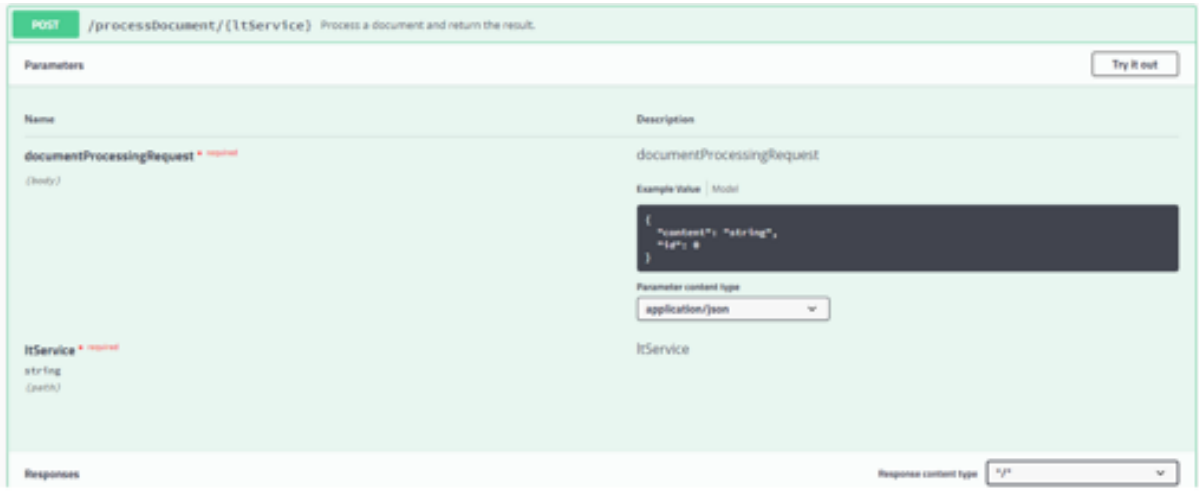

Figure 8: Swagger-based documentation for execution REST API

In the future we plan to extend the orchestrator by:

- Offering more processing options; the respective web services at LT service execution orchestrator will be added. For example, we plan to add a service that enables the processing of datasets that are described and hosted in the ELG catalogue.
- Integrating it with the ELG generic storage solution, which either uses a local disk or a cloud solution such as S3.
- Adding authentication and authorization mechanisms so that only authenticated users can use the REST API.
- Adding mechanisms for handling failures (on the processing backend).

In addition, we plan to add the option of calling/using an LT service that is deployed outside of the dedicated ELG base infrastructure (SysEleven cluster). For this task, we will exploit our experience from the OpenMinTeD project, where a web service specification was created for that reason[34]; however, we also have to take into account the LT service internal APIs (T2.5).

[33] https://swagger.io/
[34] https://openminted.github.io/releases/processing-web-services/1.0.0/specification

9.3 LT processing services registration

Currently, the process of integrating a new containerized LT tool/service is manual: a Docker image is prepared by the respective provider; the image contains the required operating system and software dependencies. The LT service that is packaged includes code for (a) interacting with the messaging middleware (e.g., RabbitMQ) and (b) consuming/generating the required LT internal API messages. A deployment config file for k8s that points to the provided image is prepared and executed based on the Helm package manager; for more information see Section 5. Then we test whether the deployed application is able to receive messages posted by the LT service orchestrator and appropriately respond. This "trial and error" process is continued until the LT tool/service is correctly integrated to the platform.

In the future we plan to automate this procedure to the extent possible, implementing the following steps:

- Template, example code and instructions are publicly provided to help developers wrap their LT tools/services. Validators (e.g., scripts) are also provided for checking that the wrapped tool/service is able to consume request and write the respective responses.
- We create web pages in the catalogue for registering an LT service; there some minimal metadata will be provided e.g., a pointer to the Docker image.
- The registered service will not directly be published to the ELG catalogue. An administrator will receive the registration request and will run some tests offline. The approval of the LT service by the administrator will trigger publishing to the catalogue, following the lifecycle described in Section 8.3, i.e., internal → ingested → published.

10 ELG platform management and support services

Apart from the ELG basic/core services (e.g., catalogue, CMS and LT services execution) there are also modules that (a) manage users and control access, (b) monitor the whole software stack, (c) gather/generate reports (analytics) for the usage of the ELG platform and resources, and (d) facilitate licensing and billling. These modules are presented in the sections that follow.

10.1 User management

The user management module of the platform enables ELG administrators to manage user access to various resources and operations of the ELG platform and includes two main submodules: registration and authentication/authorization.

10.1.1 Registration

Users may register at the platform by providing a set of mandatory information (at least username, password and email). A registration confirmation mechanism, which requires the user to respond to a "Confirm registration" email sent after successful registration to verify the email address and activate the account, is planned in order to prevent bogus registration requests.

10.1.2 Authentication/Authorization

The authentication process is based on username-password pairs; i.e., as usual, a user is asked for a username-password which is sent to the server and compared with the ones stored in the database. An authenticated user is provided with a JSON Web Token with a predefined expiration date. JSON Web Tokens (JWT) is an open,

industry standard RFC 7519[35] method for representing claims securely between two parties (users, servers, or any other combination of services). The JWT, in the case of ELG, is used between the user's web browser and the backend services to ensure that only authenticated users can interact with the platform when this is required. Security of the communication between the parties is ensured by digitally signing JWTs with an encryption algorithm, so the receiving party can trust the transmitted information. JWTs are compact and small enough to be sent through a POST request in an HTTP Header, encompassing information that the receiving party needs (e.g., username, user permissions, token expiration date, etc.). Apart from the ELG platform registered users, we are also exploring the option (if it is required) to give access to single sign-on (SSO) users (e.g., Shibboleth) who are authenticated by external identity providers (IdPs).

User privileges or access levels related to ELG's resources and services are handled by the authorization submodule. Authorization is based on user roles and groups, assuming specific access permissions to operations like resource creation/update/retrieval/deletion/download or service execution. The main user roles include "administrators" and "editors" (Figure 9). User roles can be extended and refined in order to accommodate the needs of the different user types as described in Section 2. Furthermore, defining editor groups will also be possible. Users belonging to an editor group, managed by appointed group managers, can access all resources created by the group, if these resources are in **published** or **ingested** status (**internal** resources are visible only to their creator(s)).

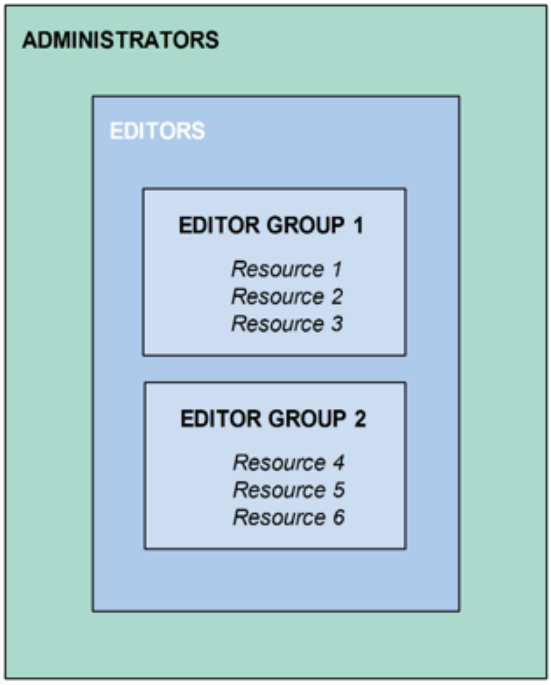

Figure 9: Basic user roles, user groups and permission scopes in the ELG platform

[35] https://tools.ietf.org/html/rfc7519.html

10.2 Monitoring

For monitoring the ELG platform, we can use the cAdvisor (container Advisor)[36] agent. cAdvisor is installed by default on all kubernetes cluster nodes and it collects metrics for CPU, memory, filesystem, and network usage for all containers that run in the node. In order to gather these metrics/statistics from the cAdvisor agents the Prometheus system can be used[37]. Prometheus provides (a) a query language and a dashboard for querying and visualizing the gathered metrics and (b) mechanisms for defining alert conditions and sending the respective notifications; the aforementioned features are very useful in monitoring the whole ELG platform.

10.3 Analytics

Analytics derived from data, such as logs, resource views, downloads, LT service executions request/jobs or any other information that is stored in the database can be created based on the Elasticsearch engine, by extending its usage with Logstash and Kibana, the main components of the Elastic Stack[38]. Logstash is a system for ingesting data (e.g logs), transforming them and finally indexing them in Elastic, and Kibana a data visualization plugin for the indexed content.

10.4 Licensing, billing and payments

Licensing terms will be assigned by the resource provider, preferably with standardized licences. Free open access licences for data resources and software will be encouraged where possible. However, providers will also be able to share their content on a charge base. Pre-defined billing schemes are currently under discussion looking at the various possibilities of offering content: e.g., once-off payment for downloading resources, subscription-based billing, billing per quota/volume usage or running instances, discounts for academic users or non-commercial use, etc.

In order to provide access to an LT service or LR according to its licensing terms, we are investigating the use of a licence server and/or an ELG-specific licence management component. A licence server controls which and how many instances of a software are allowed to run; in ELG, this server can be used to control how many containers of an LT tool can be used/run within our k8s cluster. The same functionalities can be provided by an ELG-specific software by exploiting information that is given from the LT service provider, such as number of replicas allowed, etc. The selection of the final option will take into consideration solutions that can be easily installed/configured and integrated with the rest of the platform.

11 User interface

The user interface for the catalogue, the admin pages, etc. is created using Angular (version >=7), a popular javascript framework. For creating/rendering the respective HTML web pages, the backend REST services are used (e.g., catalogue REST API) to get the required information in JSON format.

For the Content Management System (CMS) the Drupal package has been selected. A Docker image for Drupal has already been created and used for deploying it to our development k8s cluster.

---

[36] https://kubernetes.io/docs/tasks/debug-application-cluster/resource-usage-monitoring/
[37] https://prometheus.io/
[38] https://www.elastic.co/products/

The design of the UI layer and the technologies used will be described in detail in the respective deliverables of WP3, D3.1 and its updates (D3.2, D3.3. and D3.4).

12 Nginx server as a gateway

All publicly available ELG services are served/proxied via an Nginx web server (an ingress controller). For each service (e.g., CMS, LT service execution, catalogue) that is exposed to the internet an appropriate k8s config file is created that specifies the mapping between a publicly accessible URL/endpoint to the respective backend service name (and port) deployed at the kubernetes cluster. The Nginx ingress controller is also deployed as container in the k8s cluster.

13 References

Broeder, Daan, Thierry Declerck, Erhard Hinrichs, Stelios Piperidis, Laurent Romary, Nicoletta Calzolari, et al. (2008) "Foundation of a Component-Based Flexible Registry for Language Resources and Technology". In *Proceedings of the 6th International Conference of Language Resources and Evaluation (LREC 2008)*. European Language Resources Association (ELRA) <http://www.lrec-conf.org/proceedings/lrec2008/pdf/364_paper.pdf>

Gavrilidou, Maria, Penny Labropoulou, Elina Desipri, Stelios Piperidis, Haris Papageorgiou, Monica Monachini, et al. (2012) "The META-SHARE Metadata Schema for the Description of Language Resources". In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC2012)*, Istanbul, Turkey. European Language Resources Association (ELRA) <http://www.lrec-conf.org/proceedings/lrec2012/pdf/998_Paper.pdf>

Ide, Nancy, Keith Suderman, James Pustejovsky, Eric Nyberg, Christopher Cieri, and Marc Verhagen (2016) "The Language Application Grid and Galaxy". In *Proceedings of the Tenth International Conference on Language Resourcesand Evaluation (LREC 2016)*, Portorož, Slovenia. European Language Resources Association (ELRA)

Jörg, Brigitte, Hans Uszkoreit and Alastair Burt (2010) "LT World: Ontology and Reference Information Portal". In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010),* Malta. European Language Resources Association (ELRA)

Labropoulou, Penny, Dimitris Galanis, Antonis Lempesis, Mark Greenwood, Petr Knoth, Richard Eckart de Castilho, et al. (2018) "OpenMinTeD: A platform Facilitating Text Mining of Scholarly Content". In *WOSP 2018 Workshop Proceedings, Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA) <http://lrec-conf.org/workshops/lrec2018/W24/pdf/13_W24.pdf>

McCrae, John, Penny Labropoulou, Jorge Gracia, Marta Villegas, Víctor Rodríguez-Doncel, and Philipp Cimiano (2015) "One Ontology to Bind Them All: The META-SHARE OWL Ontology for the Interoperability of Linguistic Datasets on the Web". In *The Semantic Web: ESWC 2015 Satellite Events*, ed. by Fabien Gandon, Christophe Guéret, Serena Villata, John Breslin, Catherine Faron-Zucker, and Antoine Zimmermann, Lecture Notes in Computer Science, pp. 271–82. Springer International Publishing <https://link.springer.com/chapter/10.1007/978-3-319-25639-9_42>

Okman, Lior, Nurit Gal-Oz, Yaron Gonen, Ehud Gudes and Jenny Abramov (2011). "Security Issues in NoSQL Databases". 10.1109/TrustCom.2011.70.

Piperidis, Stelios, Penny Labropoulou, Miltos Deligiannis, and Maria Giagkou (2018) "Managing Public Sector Data for Multilingual Applications Development". In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA) <http://www.lrec-conf.org/proceedings/lrec2018/pdf/648.pdf>

Wilkinson, Mark D., Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, et al. (2016) "The FAIR Guiding Principles for Scientific Data Management and Stewardship". *Scientific Data*, 3, 160018 <https://doi.org/10.1038/sdata.2016.18>