

# EUROPEAN LANGUAGE GRID

D2.6

ELG platform (final  
release)

---

Authors:

Penny Labropoulou (ILSP), Dimitris Galanis (ILSP), Miltos Deligiannis (ILSP), Katerina Gkirtzou (ILSP), Leon Voukoutis (ILSP), Athanasia Kolovou (ILSP), Dimitris Gkoumas (ILSP), Juli Bakagianni (ILSP), Stelios Piperidis (ILSP), Florian Kintzel (DFKI), Remi Calizzano (DFKI), Georg Rehm (DFKI), Ian Roberts (USFD), Kalina Bontcheva (USFD), Andis Lagzdīņš (TILDE), Aivars Bērziņš (TILDE)

Dissemination Level:

Public

Date:

28-02-2022

## About this document

Project	ELG – European Language Grid
Grant agreement no.	825627 – Horizon 2020, ICT 2018-2020 – Innovation Action
Coordinator	Prof. Dr. Georg Rehm (DFKI)
Start date, duration	01-01-2019, 42 months (GA amendment version: AMD-825627-7)
Deliverable number	D2.6
Deliverable title	ELG platform (final release)
Type	Report
Number of pages	52
Status and version	Final – Version 1.0
Dissemination level	Public
Date of delivery	Contractual: 28-02-2022 – Actual: 28-02-2022
WP number and title	WP2: Grid Platform – Language Grid
Task number and title	Task 2.3: Implementation, integration and initial population of the ELG platform; Task 2.4: Development and integration of APIs between technical components of the core ELG system; Task 2.7: Operations, maintenance, updates – upload and integration of new grid content; Task 2.8: Development and integration of ELG platform management and maintenance
Authors	Penny Labropoulou (ILSP), Dimitris Galanis (ILSP), Miltos Deligiannis (ILSP), Katerina Gkirtzou (ILSP), Leon Voukoutis (ILSP), Athanasia Kolovou (ILSP), Dimitris Gkoumas (ILSP), Juli Bakagianni (ILSP), Stelios Piperidis (ILSP), Florian Kintzel (DFKI), Remi Calizzano (DFKI), Georg Rehm (DFKI), Ian Roberts (USFD), Kalina Bontcheva (USFD), Andis Lagzdīņš (TILDE), Aivars Bērziņš (TILDE)
Reviewers	Katrin Marheinecke (DFKI), Andres Garcia Silva (EXPSYS)
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 HENSOLDT Analytics GmbH (HENS), Austria Expert System Iberia SL (EXPSYS), Spain University of Edinburgh (UEDIN), United Kingdom
EC project officers	Philippe Gelin, Miklos Druskoczi
For copies of reports and other ELG-related information, please contact:	DFKI GmbH European Language Grid (ELG) Alt-Moabit 91c D-10559 Berlin Germany Prof. Dr. Georg Rehm, DFKI GmbH georg.rehm@dfki.de Phone: +49 (0)30 23895-1833 Fax: +49 (0)30 23895-1810  <a href="http://european-language-grid.eu">http://european-language-grid.eu</a> © 2022 ELG Consortium

## Table of Contents

List of Figures	4
List of Tables	4
List of Abbreviations and Acronyms	5
Abstract	7
1 Introduction	7
2 The ELG Platform	9
2.1 Overview	9
2.2 ELG Platform Architecture	9
2.3 Development Roadmap	10
3 ELG Base Infrastructure	11
3.1 Kubernetes Clusters	11
3.2 Management of Source Code Docker Images and LT services	12
3.3 Storage	13
4 Background: Metadata Model, User Model and Publication Policies	14
4.1 Metadata Model	14
4.1.1 Overview of the Model	14
4.1.2 Full, Minimal and Relaxed Versions of the Model	17
4.2 Publication Lifecycle of Metadata Records and Publication Policies	18
4.3 User Categories	21
5 ELG Platform Backend	22
5.1 ELG Catalogue	22
5.1.1 Catalogue Application	22
5.1.2 Database	24
5.1.3 Indexing and Search Components	24
5.1.4 Components and Procedure for the Upload/Download and Storage of Content Files	25
5.1.5 Registration and Publication of ELG-compliant LT Services	26
5.1.6 Catalogue Population	27
5.1.7 Assignment of Persistent Identifiers	30
5.1.8 Export of Metadata Records and Exposure through Other Catalogues	31
5.2 LT Processing Services Execution Backend	31
5.2.1 Internal LT APIs	31
5.2.2 Kubernetes and Knative	33
5.2.3 LT Service Execution Server and External LT API	34
5.2.4 LT Service Helper Services	36
5.3 ELG Platform Management and Support Services	36
5.3.1 User Management	36
5.3.2 Monitoring and Analytics	37
5.3.3 Licensing and Billing Module	39
6 ELG Platform Access Methods	41
6.1 Access through the Website	41
6.1.1 Catalogue User Interface	42

6.1.2	Integration with the Website	48
6.2	Python SDK Toolkit	48
6.2.1	Browsing the Catalogue	48
6.2.2	Interaction with the Resources	49
7	Conclusions	51
8	References	51

## List of Figures

Figure 1:	ELG user manual	8
Figure 2:	ELG architecture	10
Figure 3:	ELG-compatible service with components running outside ELG	13
Figure 4:	Overview of the ELG-SHARE entities	15
Figure 5:	Excerpt of the ELG metadata model (focusing upon tools/services)	16
Figure 6:	Minimal version for ELG compatible services	17
Figure 7:	Minimal version for corpora	18
Figure 8:	ELG publication lifecycle	20
Figure 9:	Claim of metadata records	29
Figure 10:	JSON input	32
Figure 11:	JSON output	32
Figure 12:	Knative configuration example	34
Figure 13:	Screenshot of the service-specific Grafana dashboard	38
Figure 14:	Billing workflow between ELG and Chargebee platforms	40
Figure 15:	Chargebee subscription management window	41
Figure 16:	Browse/Search page of the ELG catalogue	42
Figure 17:	Free text search	43
Figure 18:	Landing page of an ELG-compatible service	44
Figure 19:	Try out GUI	45
Figure 20:	Editor form for corpus	46
Figure 21:	My grid	47
Figure 22:	My items	47
Figure 23:	Python SDK Catalogue code example	49
Figure 24:	Python SDK Corpus code example	49
Figure 25:	Python SDK Service code example	50
Figure 26:	Screenshot of the local installation GUI	51

## List of Tables

Table 1:	ELG platform release plan	11
Table 2:	Validation types by source and type of metadata record	21

## List of Abbreviations and Acronyms

API	Application Programming Interface
ASR	Automatic Speech Recognition
CD	Continuous Deployment
CI	Continuous Integration
CMS	Content Management System
CPU	Central Processing Unit
CRUD	Create, Read, Update and Delete
DoA	Description of Action
DOI	Digital Object Identifier
ELG	European Language Grid
FAIR	Findable, Accessible, Interoperable, Reusable
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IE	Information Extraction
IRI	Internationalised Resource Identifier
JSON	JavaScript Object Notation
JSON-LD	JSON – Linked Data
JWT	JSON Web Token
K8s	Kubernetes
LOD	Linked Open Data
LR	Language Resource
LRT	Language Resource and Technology
LT	Language Technology
MIME	Multipurpose Internet Mail Extensions
MT	Machine Translation
MVP	Minimum Viable Product
NCC	National Competence Centre
NLP	Natural Language Processing
PID	Persistent Identifier
R1	Release 1
R2	Release 2
R3	Release 3
REST	Representational State Transfer
SDK	Software Development Kit
SPA	Single Page Application
TC	Text Classification
TTS	Text to Speech Synthesis
UI	User Interface
UML	Unified Modeling Language

URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
XML	Extensible Markup Language
XSD	XML Schema Definition
YAML	YAML Ain't Markup Language

## Abstract

This document introduces the final release (Release 3, R3) of the European Language Grid (ELG) platform. We present the platform's architecture, its main building blocks, components and functionalities as well as the software and application frameworks used. The document includes an overview of the base infrastructure, the platform backend (with a detailed presentation of the catalogue application and the execution component for the LT processing services), and a short account of the catalogue frontend. A detailed presentation of the ELG catalogue graphical user interface is included in D3.4. The current contents of the catalogue and the ELG metadata model are also briefly described. Finally, the ELG user manual, an online document serving all ELG platform users (consumers, providers, validators of Language Resources and Technologies and administrators of the ELG Platform), which was initiated in Release 1, has been updated to reflect the current changes and additions. ELG Release 3 is available at <https://live.european-language-grid.eu>.

## 1 Introduction

Deliverable D2.6 consists of two parts:

- the **software** of the final release (Release 3, R3) of the European Language Grid (ELG) platform<sup>1</sup>, i.e., the modules implementing the main functionalities foreseen for this release, and the current platform contents, namely a set of ready-to-deploy Language Technology (LT) services, a set of Language Resources (LRs) available for direct download, as well as metadata descriptions for Language Resources and Technologies (LRTs) and LT-related actors and activities;
- this **report**, which provides an overview of the platform, its architectural design and main components, the software and application frameworks used for their implementation, the application programming interfaces (APIs), as well as the operations and supported user interactions.

The final release includes the main building blocks required for the operation of the platform:

- the **user management** component with all user roles defined for ELG,
- the components that support the **uploading, documenting, storing, managing, and downloading of resources** with their documentation (metadata records),
- the components that support the documenting and managing of metadata records,
- the components that enable the **browse and search** features of the **catalogue** and expose the metadata records to users,
- the components that are responsible for the **execution of the LT services**,
- the **APIs** required for interacting with other layers (front-end, execution of LT services),
- the prototype **billing** component, and
- the component for **assigning Digital Object Identifiers (DOIs)** to ELG-hosted resources and services<sup>2</sup>.

This report is organised as follows: Section 2 provides an overview of the ELG platform, its architecture, and the development roadmap. Section 3 presents the base infrastructure on which the platform is deployed. Section 4

---

<sup>1</sup> The platform code is maintained at <https://gitlab.com/european-language-grid/platform>, as it is under constant development throughout the project duration. We currently discuss the appropriate software licence to apply to the code base, so that it can be shared with an open permissive licence enabling contributions from third parties.

<sup>2</sup> The implementation and integration of this component is under way; for more information, see Section 5.3.4.

presents the models and policies that influence the design and operations of the ELG platform; these include the metadata model, the publication lifecycle of metadata records, and the user management model. Section 5 focuses on the ELG platform backend, which 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. Section 6 presents the components that support access to the ELG Platform, i.e., the frontend and the Python SDK.

Finally, the ELG user manual (Figure 1), with instructions for all users (content on consumers, providers and administrators) of the ELG platform<sup>3</sup>, forms an integral part of this deliverable, albeit as an online document at <https://european-language-grid.readthedocs.io>. At the time of writing, it includes the following contents:

- introduction and overview of the catalogue contents
- instructions for consumers (search and view of the catalogue and catalogue items, testing of LT services and download of resources)
- instructions for providers, with subsections for ELG-integrated LT services (LT tools and services ready to be deployed in the platform) and data resources (i.e., corpora, models, lexica, terminologies, etc.); step-by-step instructions for the creation and management of metadata records, with examples and links to additional material are included in this section
- instructions for validators & administrators
- annexes with useful links to more detailed information material (e.g., the specifications of the ELG API for LT service execution, instructions on containerisation, etc.) and the use of the Python SDK.

Given that the implementation of the ELG platform is ongoing, the manual is conceived as a living document that we continuously update following the evolution of the platform.

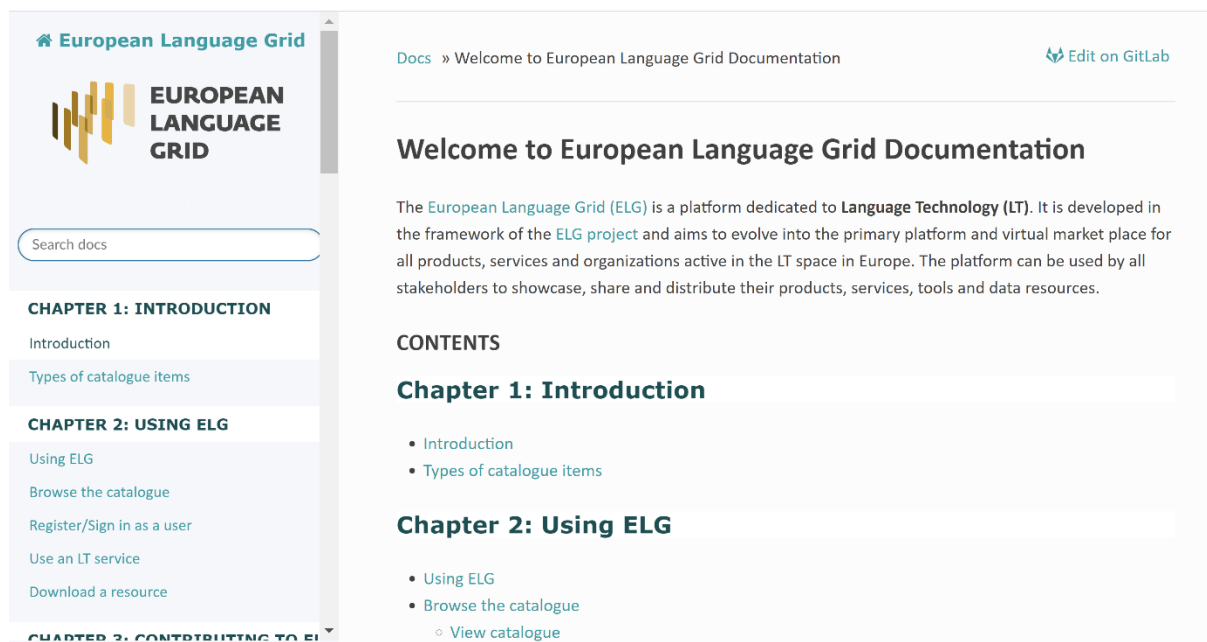


Figure 1: ELG user manual

<sup>3</sup> For information on types of users, see Sections 2.1 and 4.3, and Deliverables D2.1 and D3.1.



## 2 The ELG Platform

The following section introduces the ELG platform, the platform architecture and the development roadmap, giving an overview of the main features of each release.

### 2.1 Overview

The European Language Grid (ELG) platform [Rehm et al. 2020] aims to become the primary platform for Language Technology (LT) in Europe; it is developed to be a scalable cloud infrastructure that provides access to commercial and non-commercial LTs for all European languages, including LT tools/services deployed within ELG, and data resources (datasets, lexica, models, etc.). The European LT community can upload their technologies and datasets into ELG in an easy and efficient way, deploy them through the Grid, and connect them with other resources. In addition, the ELG platform offers information for and about the LT domain and activities, such as information on projects and stakeholders.

The ELG project and platform targets a wide range of users with different requirements and expectations<sup>4</sup> (cf. Deliverables D2.1, D3.1 and D7.1 for more information):

- **content providers** and **developers** and **integrators**, i.e., providers and consumers of LT resources,
- **information providers** and **information seekers**, i.e., providers and consumers of LT-related (meta)-information,
- **citizens**, i.e., individuals that are interested in LT (essentially a subset of information consumers),
- **ELG platform and content technical users and validators**, i.e., the ELG technical team that maintains and monitors the day-to-day operation and performance of the platform and performs operations related to the platform contents.

Users may identify themselves with one or more of the above categories when interacting with the system, with different needs each time. In the rest of the document, we refer to them as *consumers*, *providers*, *validators* and *administrators*.

ELG R3 extends and improves the functionalities introduced in previous releases. The focus for this release has been on

- the interaction with other infrastructures and catalogues, by facilitating the import of metadata records from other catalogues and exposing metadata records to other catalogues,
- enhancing the support of the publication lifecycle, through the extension of validation checks and the development of GUI forms for the validation and service registration process,
- the improved support of the versioning of metadata records and LT services,
- the design and development of a prototype billing model, and
- the setup of an operational module for the assignment of persistent identifiers.

### 2.2 ELG Platform Architecture

An overview of the ELG platform architecture is shown in Figure 2. The platform consists of three main layers: the **base infrastructure**, the **platform backend** and the **platform frontend** (user interface).

---

<sup>4</sup> See Deliverables D2.1, D3.1 and D7.1 for more information.

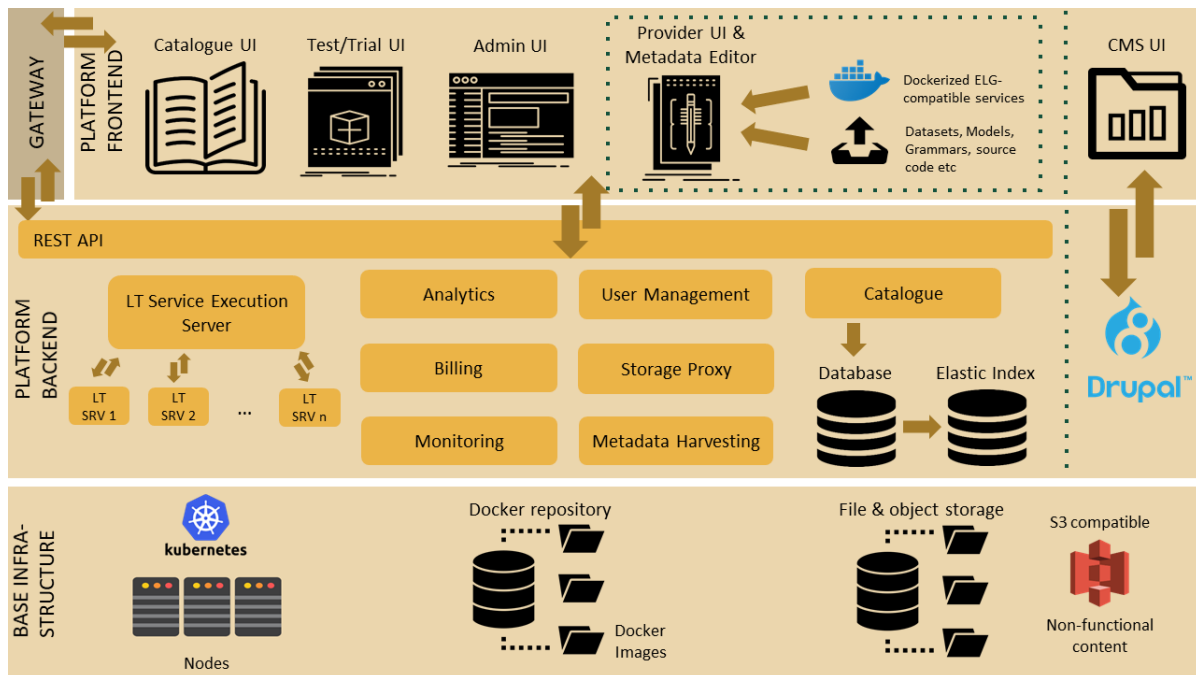


Figure 2: ELG architecture

The **base infrastructure** (Section 3) is the layer on which all ELG software components are deployed and run; it includes the supporting tools that facilitate development and management of the ELG platform software.

The **platform backend** layer (Section 5) consists of all the components that enable the operation of the ELG platform, i.e., the core components (such as the catalogue database and index), the component for processing LT services and the platform support and management components (e.g., the user management module).

The **platform frontend** layer (Section 6) consists of (a) the static pages maintained in the Content Management System (CMS), which aim to provide information on the project and the LT domain and activities, and (b) the platform user interfaces including the catalogue UI (i.e., the browse/search page with all the metadata records, the view pages of the metadata records), the ELG interactive metadata editor for registering resources, the user dashboard, etc. The catalogue UI consumes REST services exposed by the ELG platform backend (e.g., catalogue application, LT Service execution server).

This report focuses on the platform backend, i.e., the platform components, the ELG catalogue, the LT service execution layer and the supporting components and functionalities. For the sake of completeness, an abridged description of the frontend components supporting the interaction of users with the ELG platform is also included here, while the CMS and its contents are described in Deliverables D3.2 – D3.4.

## 2.3 Development Roadmap

As specified in the amended ELG Description of Action (DoA), the platform is developed and delivered in three major releases in April 2020 (M16), March 2021 (M27) and February 2022 (M38) (cf. Table 1).

Two additional pre-releases had been made available before R1: (1) a demonstration version (Minimum Viable Product version, MVP), developed at an early stage and presented in October 2019 (M10 of the project) at META-FORUM 2019, i.e., the first Annual ELG Conference (see D7.5); (2) an alpha release made available on request to interested users – especially those who were preparing proposals for the ELG Open Call – in February 2020 (M14).

<b>ELG platform Release 1 (R1)</b> (D2.4; delivered M16)	<ul style="list-style-type: none"> <li>Backend components required for the operation of the catalogue:             <ul style="list-style-type: none"> <li>simple user management component,</li> <li>components supporting documentation/uploading, storing/downloading of all resource types (tools and services, datasets, etc.), z&lt;</li> <li>APIs required for interacting with other layers</li> </ul> </li> <li>First version of the guidelines on its use and provision of resources, instructions for containerisation and invoking remotely accessible web services</li> <li>Limited sets of tools and services and LR</li> </ul>
<b>ELG platform Release 2 (R2)</b> (D2.5; delivered M27)	<ul style="list-style-type: none"> <li>Updated version of the platform including the components and APIs required for running language processing services (containerised services stored in the ELG and web services via REST APIs) directly in ELG</li> <li>Updated version of the guidelines on its use and provision of resources, instructions for containerisation and invoking remotely accessible web services</li> <li>Updated catalogue with resources from ELG partners</li> </ul>
<b>ELG platform Release 3 (R3)</b> (D2.6; due M38)	<ul style="list-style-type: none"> <li>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</li> <li>Updated catalogue with resources from ELG pilots and collaborating initiatives</li> </ul>

Table 1: ELG platform release plan

## 3 ELG Base Infrastructure

The base infrastructure is the layer on which the ELG platform (backend and UI) runs. It also includes tools for software development and consists of the following parts: two **Kubernetes<sup>5</sup> clusters** (one for development and a second one for production), **source code repositories**, **external/internal container registries<sup>6</sup>**, and a **file and object storage module**. More detailed information regarding the ELG Base Infrastructure can be found in the dedicated infrastructure deliverables (D1.1 – D1.5), while this section gives an overview of the main features.

### 3.1 Kubernetes Clusters

The ELG platform software can be divided into two parts:

- the core components, e.g., the catalogue database, the metadata index, the catalogue application, etc., which are developed and maintained exclusively by the ELG technical team, and
- the set of LT tools/services that are integrated into ELG, i.e., made available through the ELG platform, and that have been developed by consortium partners or external providers.

All components and services mentioned above run in ELG as Docker containers in a Kubernetes (in short k8s) cluster. This facilitates management and deployment (see D1.5 and D2.2). The components that run in the cluster are grouped in k8s namespaces based on their functionalities. A k8s namespace is a virtual sub-cluster, which can be used to restrict access to the respective containers that run within it. For instance, we have an

<sup>5</sup> <https://kubernetes.io>

<sup>6</sup> <https://www.docker.com>

"elg-core" namespace for core components such as Keycloak<sup>7</sup>, which manages user authentication and nginx<sup>8</sup>, the web server that acts as gateway/proxy. In the "elg-backend" namespace, we have deployed the database, the indexer and the REST-based catalogue application. LT services are deployed by default in the "elg-srv" namespace; however, separate dedicated namespaces are allocated for LT services for which restricted access is required.

Currently, ELG uses two clusters: the first cluster hosts the production instance<sup>9</sup> of the platform, while the second one<sup>10</sup> (which is deployed on more limited hardware resources) is used for development and testing. Both clusters run on Virtual Machines (VMs)/hardware of SysEleven<sup>11</sup>, a Berlin-based cloud service provider. SysEleven is ELG's subcontractor and was chosen in a selection process at the beginning of the project (see D1.1).

Deployment of an application in a Kubernetes cluster requires a set of configuration files in YAML<sup>12</sup> format that specify the required information, such as the Docker image location, the number of replicas that will be deployed, the ports that will be exposed to the rest of the cluster, and the deployed service's name. The latter can be used for access by other k8s applications/services. 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, deployments in SysEleven. For these we use Helm<sup>13</sup>, a k8s package manager that automates deployments. Helm uses (generic) templates for describing k8s resources. For each different installation/deployment, the templates are filled with the respective configuration data by running a script. The generated configuration files are then submitted to the cluster via the respective k8s API. The required scripts, helm templates and configuration files that instantiate a specific ELG deployment are kept in a separate branch in ELG's GitLab repository<sup>14</sup>, i.e., we have separate branches for production and development. For automating deployments, a continuous integration/continuous deployment (CI/CD) pipeline was created. When a source file in a branch is changed, the CI/CD pipeline is triggered. It downloads the latest helm templates and scripts from GitLab and runs the scripts that inject the configuration. Finally, it deploys the final resource files to the respective k8s cluster.

All ELG services (e.g., CMS, LT service execution, catalogue backend and UI) are exposed to the public internet via a Nginx web server (an ingress controller). For each service the appropriate code is inserted to the respective k8s config files that specifies the mapping between a publicly accessible URL/endpoint to the respective internal k8s backend service name (and port). Nginx is also deployed as a Docker container.

### 3.2 Management of Source Code Docker Images and LT services

The platform's source code is hosted in an "organisation" account on GitLab (<https://gitlab.com/european-language-grid>), a code repository and development platform (for Git projects), similar to GitHub and BitBucket. All code stored within the ELG organisation account on GitLab is maintained by the ELG consortium which ensures proper versioning of all artifacts. GitLab was chosen because it provides numerous features for free, such as an unlimited number of private and public repositories, a built-in Container registry for storing images and built-in CI/CD functionalities; these CI/CD pipelines are different from the ones described in Section 3.1, which are used

---

<sup>7</sup> <https://www.Keycloak.org>

<sup>8</sup> <https://www.nginx.com>

<sup>9</sup> <https://live.european-language-grid.eu>

<sup>10</sup> The ELG development cluster is accessible to consortium members only by using IP whitelisting.

<sup>11</sup> <https://www.syseleven.de/en/>

<sup>12</sup> <https://yaml.org>

<sup>13</sup> <https://helm.sh>

<sup>14</sup> <https://gitlab.com/european-language-grid>

for deploying to our k8s clusters. The GitLab CI/CD is used, for example, to create the images that host the ELG core components as well as images for LT services. In addition to GitLab's Container registry, we make use of other registries, such as DockerHub<sup>15</sup> or Azure Container Registry<sup>16</sup> for pulling components (e.g., PostgreSQL) or LT services deployed at ELG.

The LT services that are deployed at ELG are created and maintained by different developers, groups, or companies and hosted at different registries. Thus, there are cases where ELG cannot guarantee the availability of a given LT service, for example:

1. The LT images are hosted in external container registries which are not immutable and their retention policy is controlled by their respective owner. Thus, an LT image may be altered or become unavailable.
2. Availability of a given service is restricted by specific limitations, such as relying on an external proprietary licence.
3. The availability of services which in turn call other LT services not hosted on the ELG infrastructure<sup>17</sup>.

To mitigate scenario 1, we have put a solution in place that combines the deployment of a private/internal container registry for storing the images and a process for keeping backup of the services' images. The internal container registry is deployed within SysEleven. The LT services that are deployed on the platform are pulled from their respective external original location (as defined by the service provider) on the first deployment. These images are then copied in the internal container which is used exclusively by the ELG project. Images are persisted there because (a) this registry is located nearer to the actual infrastructure and is faster to access in case images need to be pulled again and (b) because with external registries, there is no guarantee images will not be changed/overridden or remain accessible for longer periods of time. So, to be able to serve LT services consistently, all their images are copied to the dedicated ELG registry.

Use cases 2 and 3 can be detected through the automatic testing of LT services that is already in place on the ELG, but they will need to be addressed by the individual LT service providers. Regarding point 3 in particular, in R3 the catalogue UI indicates via a flag (Figure 3) whether all of the service components are hosted directly within the ELG-controlled infrastructure or whether the service relies on externally hosted services and may be subject to failures that are beyond the control of the ELG.

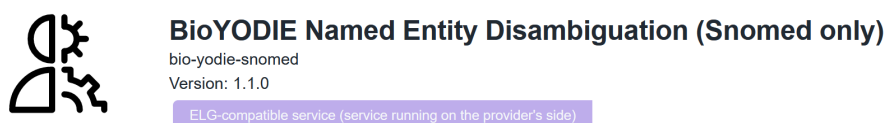


Figure 3: ELG-compatible service with components running outside ELG

### 3.3 Storage

For the storage of physical files, e.g., language resources and software delivered as source code, etc., we use an S3-compatible Object Storage solution that is hosted by SysEleven. S3-compatible Object Storage provides

<sup>15</sup> <https://hub.docker.com>

<sup>16</sup> <https://azure.microsoft.com/en-us/services/container-registry/>

<sup>17</sup> See Section 5.2 on the integration options for LT services.

functionalities that facilitate access to, and management of the data, organised in "buckets" with different access policies, thus ensuring access control and security. In the current implementation we have created a "public" and a "private" bucket. In the "public" bucket, ELG keeps all publicly accessible files (e.g., logos of organisations); no authentication/authorisation is required for them. In the "private" bucket ELG keeps files (e.g., the content files of a resource) which are not accessible to all users; i.e., their download is constrained by the licensing terms of the resource and the publication status (see Section 5.1.4 for more details).

Access restrictions to an S3 object (e.g., in the case of private buckets) are applied based on the owner of the S3 account; only via this account is it possible to control the creation and management of storage buckets, to provide temporary access to requested objects if and when needed. A user interacts with the S3 Storage through the S3 Storage Proxy and the respective authorisation module which is part of the catalogue backend. The authorisation module ensures that users have access only to the resources they own or have been granted access to. The S3 Proxy and the catalogue backend module interact with S3 based on the S3 API.

Internally within the cluster, certain parts of the platform additionally make use of network block device storage (SSD) attached to their respective containers.

## 4 Background: Metadata Model, User Model and Publication Policies

This section presents the conceptual and operational modules that influence the design of the ELG platform. We present an account of the metadata model, the publication lifecycle of ELG and the foreseen user categories.

### 4.1 Metadata Model

This subsection is devoted to the presentation of the ELG metadata model, its main features and design principles as well as the different versions that have been implemented in the ELG platform.

#### 4.1.1 Overview of the Model

The **ELG metadata model** (or ELG-SHARE<sup>18</sup>) is used for the description of all entities of interest to the ELG target users<sup>19</sup>. Essentially an application profile of the META-SHARE schema, 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 actors and activities related to LT (Figure 4).

---

<sup>18</sup> The ELG metadata model builds upon, extends and updates META-SHARE and its application profiles. Modifications have been made (a) in the contents (e.g., addition of elements for GDPR, improvement of the description of LT services and models), in response to the project requirements and more recent developments in the metadata area at large, (b) in the implementation, which now combines the XSD approach used for META-SHARE profiles with the deployment of two ontologies, namely META-SHARE (<https://w3id.org/meta-share/meta-share/>) and OMTD-SHARE (<https://w3id.org/meta-share/omtd-share/>), for the definition of the elements and values.

<sup>19</sup> For a detailed description, see Deliverable D2.3 and [Labropoulou et al. 2020].

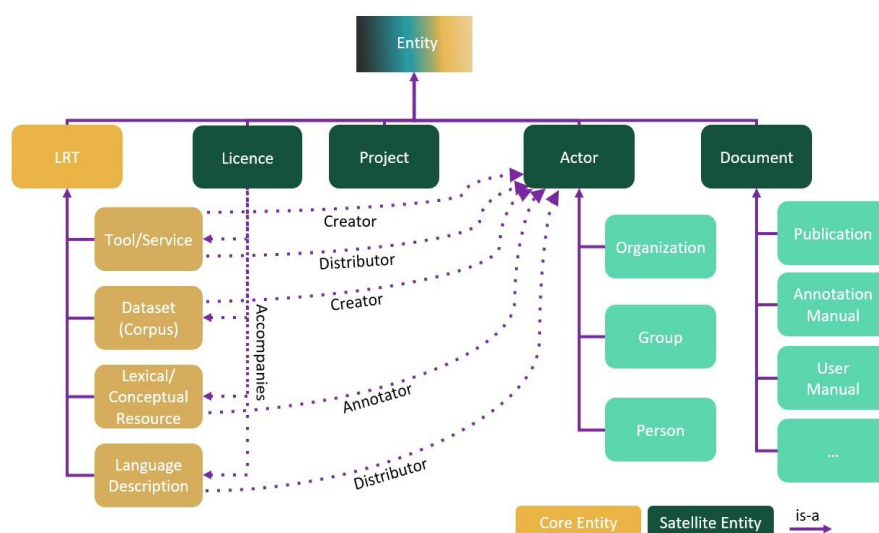


Figure 4: Overview of the ELG-SHARE entities

The model caters for the description of ELG core entities, i.e.,

- *LT tools/services*, covering all software that performs language processing and/or any LT-related operation (e.g., basic processing tools, applications, web services etc. that perform annotation, machine translation systems, speech recognisers, etc.).
- *Corpora* (datasets), defined for our purposes as structured collections of pieces of data (textual, audio, video, multimodal/multimedia, etc.), typically of considerable size and selected according to criteria external to the data (e.g., size, type of language, type of producer or expected audience, etc.) to represent as comprehensively as possible the object of study.
- *Lexical/conceptual resources*, i.e., resources (such as terminological glossaries, word lists, semantic lexica, ontologies, gazetteers, etc.) organised on the basis of lexical or conceptual units (lexical items, terms, concepts, phrases, etc.) with their supplementary information (e.g., grammatical, semantic, statistical information, etc.).
- *Language descriptions*, i.e., resources aiming to describe a language or some aspect(s) of a language via a systematic documentation of linguistic structures (e.g., computational grammars, statistical and machine learning-computed language models); it should be noted that in R3, although this class is still used in the metadata schema, the catalogue shows its subclasses, i.e., *model*, *computational grammar* and *uncategorized language description* as distinct resource types.

The ELG model also provides for entities involved in the production and usage of LTs/LRs and, in general, LT activities, i.e., *actors* (further distinguished into *organizations*, *groups*, *persons*), *documents* (e.g., user manuals, publications, etc.), *projects* and *licences/terms of use*. The model groups all metadata elements along three key concepts: *resource type*, *media type* and *distribution*. The *resource type* element distinguishes LRTs in the four classes presented above. *Media type* refers to the form/physical medium of a data resource (or of its parts, in the case of multimodal resources), i.e., text, audio, image, video and numerical text (used for biometrical, geo-spatial and other numerical data). Finally, *distribution*, following the DCAT vocabulary<sup>20</sup>, refers to the physical form of the resource that can be distributed and deployed by consumers; for instance, software resources may be distributed as web services, executable files, or source code files, while data resources may be distributed as

<sup>20</sup> <https://www.w3.org/TR/vocab-dcat-3/>

PDF, CSV, or plain text files, or through a user interface. Administrative and descriptive metadata are mostly common to all LRTs, while technical metadata differ across resource and media types as well as distributions.

Figure 5 provides an example with part of the metadata model for tools/services.

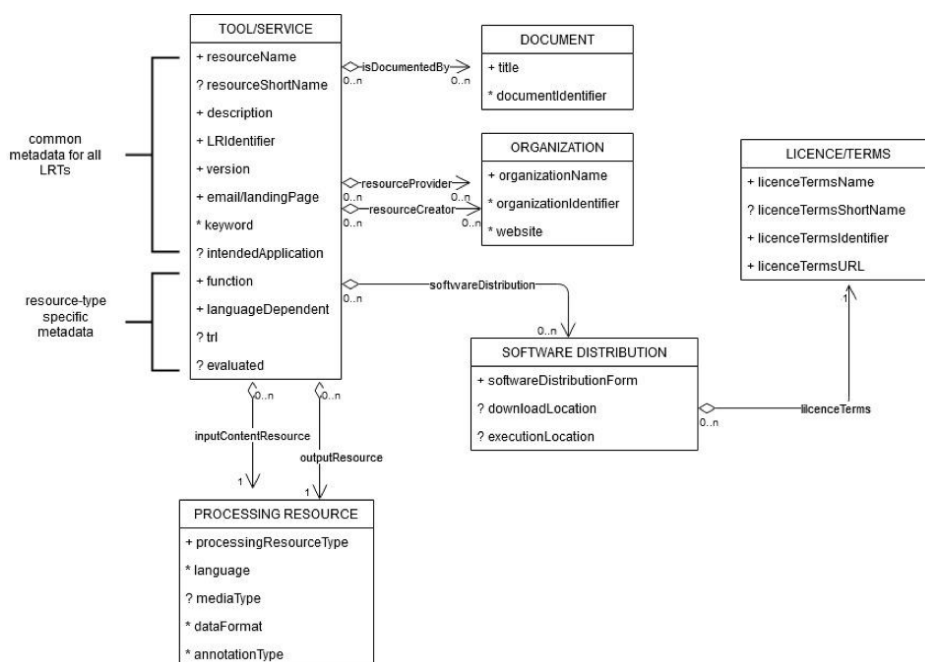


Figure 5: Excerpt of the ELG metadata model (focusing upon tools/services)

The model is implemented in the form of an XML Schema Definition (XSD)<sup>21</sup>. Its elements are linked to entities from two ontologies, namely the META-SHARE<sup>22</sup> ontology, which includes most elements and controlled vocabularies, and the OMTD-SHARE ontology<sup>23</sup>, reserved for the controlled vocabularies of LT categories (also referred to as "LT taxonomy"), data formats, annotation types and methods. Each metadata element and value has an identifier which contains the Internationalised Resource Identifier (IRI) of the corresponding entity. This approach contributes to the FAIRness<sup>24</sup> of the metadata model, facilitates linking to metadata records in other catalogues, and supports import/export in the JSON-LD serialisation format, which increases the visibility of ELG metadata records overall. The use of XSD enables us to transform the metadata schema easily into an entity-relationship model<sup>25</sup>, thus facilitating its documentation and conversion into the relational database used in the ELG catalogue backend. Ontology entities are automatically converted into XML elements, as used in the XSD. This approach enables an easy update of the schema alongside the evolution of the ontology. All relations, labels and definitions are copied into the XML elements with the same script and feed the display labels in the landing pages of the metadata records.

<sup>21</sup> The ELG metadata schema is available at <https://gitlab.com/european-language-grid/platform/ELG-SHARE-schema>, accompanied with documentation, and metadata record templates for all resource and media types.

<sup>22</sup> <https://w3id.org/meta-share/meta-share/>

<sup>23</sup> <http://w3id.org/meta-share/omtd-share/>

<sup>24</sup> The FAIR Guiding Principles for scientific data management and stewardship aim to improve the findability, accessibility, interoperability, and reuse of digital assets. The principles emphasise machine-actionability (i.e., the capacity of computational systems to find, access, interoperate, and reuse data with none or minimal human intervention) because humans increasingly rely on computational support to deal with data as a result of the increase in volume, complexity, and creation speed of data, see <https://www.go-fair.org/fair-principles/>.

<sup>25</sup> [https://en.wikipedia.org/wiki/Entity%E2%80%93relationship\\_model](https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model)



The design and the implementation of the model have been completed in R1, while updates and changes have been made throughout all the releases of the platform, taking into account user feedback, technical requirements of the platform (as its development progressed), new requirements that emerged from pilot and collaborating projects and, finally, the interoperability needs for mappings to other metadata schemas.

#### 4.1.2 Full, Minimal and Relaxed Versions of the Model

The ELG model is rich and elaborate and caters for the documentation of the full lifecycle of LRTs. The abundance of information, though, makes the task of creating metadata records quite tedious. To ensure flexibility and uptake, we distinguish metadata elements into *mandatory*, *recommended* and *optional*. The criteria used in ELG to determine the status of elements include: required for discovery, especially features considered of high interest to ELG consumers (see D2.1 and D3.1); considered indispensable for accessing the resources and, in the case of ELG-compatible services, ensuring proper deployment through the platform; supporting usage of resources; deemed valuable for experiments and projects and essential for achieving interoperability with existing metadata schemas used in the wider LT and neighbouring communities.

To be included into the ELG platform, metadata records must in principle adhere to at least the **minimal version** of the schema, i.e., they must include all the *mandatory* elements as well as the *mandatory upon condition* elements, which depend on the values of other elements. For instance, if a corpus contains personal and/or sensitive data, the provider must also encode whether these data are anonymised. Figures 6 and 7 present the minimal version of the schema for ELG-compatible services and corpora respectively; mandatory upon condition elements are indicated with an asterisk.

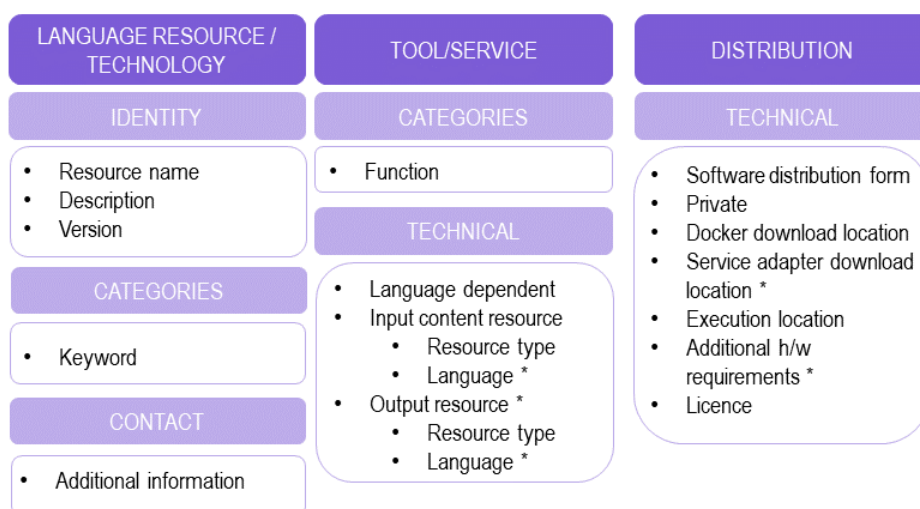


Figure 6: Minimal version for ELG compatible services

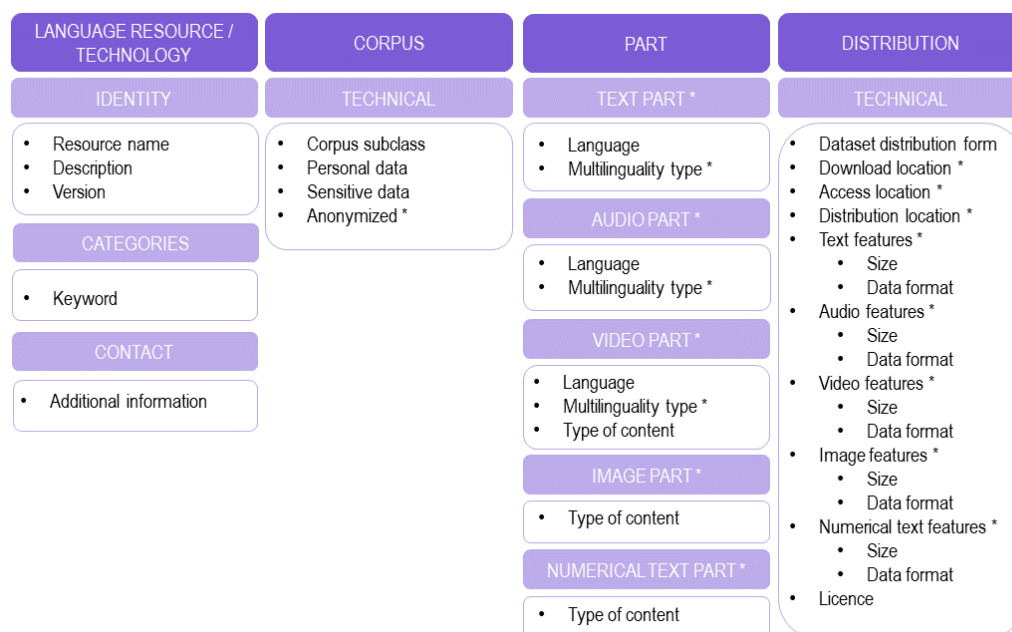


Figure 7: Minimal version for corpora

In addition, a **"relaxed" version of the schema** has been introduced in R3 that is used only for metadata records imported from catalogues with limited information or catalogues populated with metadata records of interest to a broader range of communities (e.g., Zenodo<sup>26</sup>, EOSC<sup>27</sup>, etc.) and, thus, using more general schemas (e.g. DCAT, DataCite<sup>28</sup>). This version is characterised by the following features:

- Further relaxation of the mandatoriness of specific elements and/or substitution with alternatives. For instance, the "licence" element is deemed mandatory to ensure proper (re)usability of resources<sup>29</sup>, and can only be linked to a legal document; in the relaxed version, it can be substituted with the "access rights" element, a free text element. In a similar way, the media-type specific features, for resources whose media type cannot be determined, are all attached to an "unspecified media part" component<sup>30</sup>.
- The addition of elements with free text values as an alternative to elements with controlled value vocabularies or combined elements that cannot be distinguished from the source metadata record (e.g., when size is encoded as a free text combining amount and size unit together).

## 4.2 Publication Lifecycle of Metadata Records and Publication Policies

The ELG platform can be populated from a variety of sources:

- metadata descriptions created by individuals, optionally accompanied with the respective content (physical) files (e.g. datasets). In the case of ELG-compatible LT services, an appropriate form (i.e. Docker image) of the software that allows deployment at ELG should be provided;
- metadata descriptions automatically harvested from other catalogues/repositories;

<sup>26</sup> <https://zenodo.org>

<sup>27</sup> <https://eosc.eu> and <https://eosc-portal.eu>

<sup>28</sup> <https://schema.datacite.org>

<sup>29</sup> See also principle R1.1 of FAIR principles (<https://www.go-fair.org/fair-principles/>): (Meta)data are released with a clear and accessible data usage license

<sup>30</sup> It should be noted that wherever possible, the use of one of the ELG controlled values is the preferred option. For instance, the use of media type-specific values in other elements (e.g., format values such as DOC or WAV, which are used only for text or audio files, respectively) can be used to infer the media type.

- metadata descriptions collected and converted into the ELG metadata schema with appropriate converters and imported in the ELG system in various ways; in this case, if the metadata are imported from other catalogues/repositories, the resources (data files) may remain in their source repository or be transferred to the ELG platform storage, depending on the negotiations with each source.

Depending on the source and the degree of the intended integration of the resource in the ELG platform, different requirements are set for the quality and completeness of the metadata description and the resource. A set of **publication policies and procedures** have been set in place for this objective.

The first set of policies is related to the source and mode of import of the metadata record into ELG:

- For metadata records added by individuals: only individuals who have registered in the platform and given the appropriate authorisation (i.e., assigned the provider role) can add metadata records; these can be individuals who wish to share their own LRTs or act as representatives of an organisation to which they are affiliated and wish to upload LRTs developed by that organisation. The registration procedure for users includes consent to the ELG terms of use, whereby they commit to provide resources for which they have the legal right to share them and that they are technically "safe". The provision of resources by registered users who are logged in allows for giving credit to the specific providers, but also entrusting them with all responsibility for any issues that may rise with regard to them. To further ensure quality and technical compliance to the extent possible, all metadata records added by individuals undergo a validation process, as described below.
- For metadata records automatically harvested from other catalogues: these are imported only from trusted sources following specific agreements<sup>31</sup>. Since they are already published in other repositories, and, thus, considered trustworthy, there is no reason to submit them to the human validation process.
- For metadata records collected by the ELG team and collaborating projects: This refers to
  - candidate resources identified by the ELG team in other catalogues, described semi-automatically with at least the minimal version of the ELG schema, as presented in Deliverables D5.1 – D5.3, and registered in the ELG platform through the same process envisaged for the population by individuals, hence submitted to the same publication policies;
  - bulk collection initiatives undertaken by the ELG team and a crowdsourcing initiative launched by the European Language Equality project (see Section 5.1.6.3 and D5.3). In these cases, only a subset of the required information was collected and converted to the ELG schema, i.e., the records adhere to the relaxed version of the schema; therefore, there is no reason for submitting the records to the validation process. Instead, these records are marked and can be "claimed" for further enrichment by individuals. When this happens, they follow the normal publication procedure as if they were originally submitted by individuals.

A second set of policies determines the validation types to which the metadata records are submitted based on their resource type and the degree of integration in ELG. For metadata records registered by individuals, validation at the metadata level at least is foreseen. Further requirements apply specifically to LT services in order to be integrated as ready-to-deploy in the ELG platform (see 5.1.5). For their publication, they go through a validation process that aims to ensure technical validity of the service itself. The process is performed by the ELG technical team and includes checking the technical metadata required for the import and deployment of the

---

<sup>31</sup> For more information, see Deliverables D5.1 – D5.3.

service, and the addition of ELG-specific information that will enable the execution of the service in the ELG platform. Before publishing a tool/service, the validator checks that (a) it follows the ELG technical specifications; (b) its technical metadata are as required; (c) it can be executed in the ELG platform. Similarly, for data resources uploaded in ELG, the validators make sure that the content files are conformant to the description presented in the metadata record and that they contain no malicious parts.

The **publication lifecycle** of a metadata record (Figure 8) draws upon the principles of META-SHARE and implements the above policies in the form of the following states:

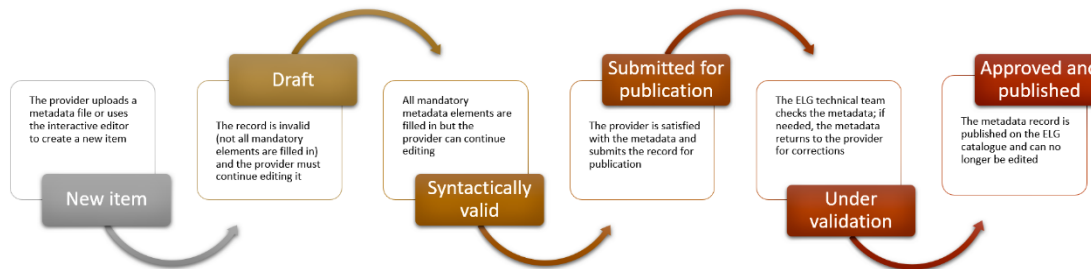


Figure 8: ELG publication lifecycle

- **new item:** A provider creates an item, by creating a metadata record through the interactive editor or by uploading a metadata file (see Section 5.1.6) and, optionally, content files (physical files with the resource contents).
- **draft:** Used as the initial state for metadata records created with the interactive editor. At this state, providers do not have to fill in all mandatory elements; only compliance as to the data type of the elements is checked (e.g., elements that take URL values must be filled in with the accepted pattern).
- **syntactically valid:** The metadata record complies with the ELG metadata schema, and all mandatory elements are filled in. The provider can still continue to edit it until satisfied with the description and can then submit it for publication.
- **submitted for publication:** As soon as the provider submits it for publication, the record becomes no longer editable and is assigned for validation.
- **under validation:** Depending on the item type and the source (see Table 2), the item is validated by designated users at the metadata, technical and legal level. The validation aims to check the consistency of the description and, where required, the technical compliance of the item to the ELG specifications; it doesn't include any qualitative evaluation. The validation is currently performed by the ELG consortium members.
- **(approved and) published:** for finalised metadata records (i.e., approved by validators); published records are available in the public catalogue.

Type of records	Validation type		
	Metadata	Technical	Legal
Harvested metadata	N/A	N/A	N/A
Metadata records uploaded by ELG admin	N/A	N/A	N/A
"Metadata only" records <sup>32</sup>	Yes	N/A	N/A
ELG compatible services	Yes	Yes	Yes
LRTs uploaded (hosted) in ELG	Yes	Yes	Yes

Table 2: Validation types by source and type of metadata record

The publication procedure for resources and the user model of ELG (Section 4.3) determines which users can access metadata records in each of the above states. For instance, providers have access to their resources at any time, but can only edit them until they submit them for publication; validators have access only to the resources they are assigned to validate, after they have been assigned to them, and only with viewing rights.

Published records cannot be changed and cannot be deleted in order to ensure reproducibility of scientific results. Only in exceptional cases can they be unpublished. Such cases mainly have to do with the reporting of a breach of Intellectual Property Rights and other legal issues, and/or technical malfunctions. If the issue is resolved, they are published again; if the issue is not resolved, they may be deleted and their landing page is replaced with a tombstone page.

Metadata records cannot be reverted to previous states, with the exception of

- records submitted for publication: when the validators find some issues, they are returned to the "syntactically valid" state in order to be editable again by the provider of the metadata records;
- published records but only in exceptional cases and only by administrators in the cases described above, as well as in the case of "claimable" metadata records that have been "claimed" by an individual; the records are then returned to the "syntactically valid" state so that they can be edited by the assigned individuals.

In addition, administrators have the right to revert all records to previous states, except for the "draft" state. Once a record is syntactically valid, it can no longer return to the draft state.

### 4.3 User Categories

The user management model comprises a set of broad user categories and roles with access policies defined in response to the functionalities of the ELG platform:

- **Unregistered users:** Users who are not registered with the platform and who are, hence, not logged in, can browse through and search the contents of the catalogue of published metadata records, and view their descriptions; they can download resources with open licences; however, they are not allowed to register a resource, or run a published LT service; they have access only to published records.
- **Registered users:** These are the users with accounts in the ELG user database. Each registered user can be assigned one or more of the following roles:

---

<sup>32</sup> "Metadata only" records are records for projects, organizations but also for LRTs that are not uploaded in ELG.

- **Consumer:** In addition to the access rights of unregistered users, they can also try out LT services with the GUIs provided in the platform, or in command mode.
- **Provider:** Providers have all the access rights of a consumer and are also able to register resources. They have editing rights for the resources they have registered until they submit them for publication.
- **Validator:** These are users who validate resources and confirm their publication. They have the same rights as a consumer and advanced rights for the resources they validate. There are three types of validators:
  - The legal validator assesses the legal standing of a language resource before it can be published.
  - The metadata validator validates the content of "metadata only" records before they are published.
  - The technical validator ensures that any ELG-hosted resource or ELG-compatible service functions as intended via the ELG infrastructure.
- **Content manager:** The team of ELG colleagues that is responsible for the smooth transition of a record submitted for publication to its published state, including the assignment of validators for ELG hosted resources and ELG compatible services; content managers have full access to all resources.
- **Administrator:** The team of ELG colleagues who develop and maintain the platform; administrators have full access to all resources and functionalities of the platform.

When users register at the ELG platform, they are automatically assigned the "consumer" role. Roles with more privileges are assigned only by technical administrators. Users can request to obtain the "provider" status through their user profile. There is no procedure for requesting the validator role since this is restricted to the ELG technical team and managed by the administrators within the consortium.

## 5 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 the following subsections.

### 5.1 ELG Catalogue

In the following subsection, we present the catalogue backend components, i.e., the catalogue application, the database, the indexing and search mechanism, the upload and download mechanism and procedure for content files stored in ELG, and, finally, discuss the population of the catalogue from the technical point of view.

#### 5.1.1 Catalogue Application

The catalogue application is a RESTful backend providing all necessary functionalities for database management and user access control to the ELG platform features. It is built with Django Web Framework (version 3.0.9) and Django REST Framework (version 3.11.1)

The catalogue application defines a set of REST API endpoints used for serving requests coming from the catalogue UI (described below in Section 6.1.1) the LT service execution server (described below in Section 5.2.3) or the command line utility (SDK) of ELG that was created using Python. The REST endpoints are controlled by the

user authorisation policies, in order to distinguish between those that are accessible by everyone (e.g., the retrieval of a published metadata record) and those that require user authentication and a specific user role (e.g., users can create metadata records if they are authenticated and have been assigned the "Provider" role).

A key functionality of the catalogue application is the management of metadata records through CRUD operations, i.e., create, read, update and delete operations, as well as supporting operations for the management of the full lifecycle of the metadata records, as described in Section 4.2. The defined REST API endpoints handle requests for:

- creating metadata records, by uploading metadata via two separate endpoints: one consuming XML and intended for batch uploads, and the other accepting JSON and used by the metadata editor UI
- updating metadata records via the interactive metadata editor; update is done only via JSON;
- retrieving a metadata record in JSON format to be rendered by the frontend (catalogue UI);
- exporting a metadata record, in XML format compliant with the ELG Schema;
- uploading a resource (content files) associated to a metadata record;
- downloading a resource in accordance with its licensing terms;
- authorising service execution (see Section 5.2.3);
- collecting usage information (see Section 5.3.2.1);
- submitting metadata records for publication;
- validating metadata records for publication.

The endpoints used for interacting with the database were created using Django REST Framework (DRF) and Django models, the latter are serialised to JSON or XML via the respective serialiser.

In addition, a REST endpoint for metadata validation is offered for the validation of metadata records against the ELG metadata schema. Providers who wish to upload metadata files, can validate their XML files via this endpoint. This functionality is publicly available for single XML files or zipped archives, for batch validation at <https://live.european-language-grid.eu/catalogue/#/validate-xml> as well as through the user's dashboard (see Section 6.1).

Finally, a REST endpoint is utilised in a functionality that aims to reduce duplicate entries and normalise values for related entities. The population of the ELG catalogue with metadata records for the LRTs includes references to other entities, such as organisations and individuals that have developed the resource, projects that have funded them, licences with which they are distributed, etc. These are often referred to with similar names or titles; e.g., the "CC-BY 4.0" licence is also encountered as "Creative Commons Attribution 4.0", "CC-BY licence", "CC BY license", etc. To tackle this, the ELG catalogue application offers a retrieval mechanism that matches records of entities already imported in the database. The criteria for the retrieval are based, in priority order, on (a) the identifiers, and (b) on the metadata elements that can serve to uniquely identify instances of each entity type (e.g., the website for organisations and the email address for persons). In addition, where available, we have pre-populated the ELG database with openly available datasets, such as the SPDX list of licences<sup>33</sup>.

---

<sup>33</sup> <https://spdx.org/licenses/>

### 5.1.2 Database

A PostgreSQL database (version 11.9) is used for the persistent storage of metadata records and of internal information required for ELG platform operations.

The database implements the ELG metadata schema in the form of a relational model. Therefore, all metadata records must comply with the ELG schema in order to be imported and saved. The catalogue backend two endpoints used for creating metadata (the one accepting JSON and the other XML) as well as the endpoint for updating metadata via JSON apply a set of validation rules that take into account the source of the record and check the record against the minimal (e.g., for records created by individuals) or relaxed version (e.g., for records from catalogues with general schemas) - see Sections 4.1, 4.2 and 5.1.6. An additional set of validation rules that takes into account the values of specific metadata elements is applied to check the syntactic and partial semantic consistency of the description (e.g., checks for "mandatory upon condition" elements). If a record does not conform to these rules, validation errors are issued and presented to the user for correction; otherwise, the records are saved in the database. The same set of validation rules are enforced by the frontend when the metadata records are created with the interactive editor. An exception is made for "draft" records (see Section 4.2) which are saved in a temporary form. In the case of "draft" records, both the backend and frontend bypass the validation of mandatory elements and validate only their datatype.

### 5.1.3 Indexing and Search Components

Published metadata records can be searched through the catalogue UI, on a subset of the metadata information indexed by Elasticsearch<sup>34</sup>, either by search queries or using faceted search. This subset includes

- for all entities, i.e., LRTs, projects and organisations: name or title, short name or title, description, keywords and source;
- in addition,
  - for organisations: country of registration, name of parent organisation (if it exists);
  - for LRTs: resource type (and, in the case of language descriptions, subtype), version, language, language variety, licence, condition of use, access rights, intended application, and whether the record is an ELG compatible service, or with data uploaded in ELG;
  - for Data Language Resources: media type, linguality type (indicating if the resource contains data in one or more languages) and multilinguality type;
  - for models: model type and model function;
  - for LT tools/services: service function and whether they are language dependent or not.

By default, the search functionality matches whole words using the OR operator. To improve search results, for the free text search functionality, the index has been enriched with synonyms of language names (retrieved from Glottolog<sup>35</sup>) and service functions (retrieved from the OMTD-SHARE<sup>36</sup> ontology). In addition, query expressions passed in the search parameter are also supported. Such expressions utilise the Lucene Query Syntax<sup>37</sup> and can be used for advanced queries such as exact phrase matches (e.g., "bilingual corpus"), fuzzy or proximity search, term boosting, etc.

---

<sup>34</sup> <https://www.elastic.co>

<sup>35</sup> <https://glottolog.org>

<sup>36</sup> <http://w3id.org/meta-share/omtd-share/>

<sup>37</sup> [https://lucene.apache.org/core/2\\_9\\_4/queryparsersyntax.html](https://lucene.apache.org/core/2_9_4/queryparsersyntax.html)



When queried by the catalogue UI component, the backend returns the search results in JSON format. The JSON contains the indexed information for the metadata records, as well as the following: creation date, last update date, number of views, number of downloads (if the metadata record has data stored in ELG), and for ELG-compatible services number of executions and whether the record is a proxied one, and whether the record is declared as "for information" or as "work in progress". The returned results can be sorted alphabetically, as well as by their last update date.

In addition, dedicated pages that serve as a focused version of the catalogue are included in the dashboard UI (see Section 6.1 and D3.3). These pages present a set of filtered records according to each user's role (e.g., for providers, a list of the items they have created, for validators, a list of the items they have been assigned for validation). They include filtered and free text search functionalities like the main catalogue page, albeit with different elements in the indices. More specifically:

- the index for the "my items" page (for providers) includes the following elements: name or title, publication status of the metadata record, item type, whether the record is accompanied by data uploaded at the ELG infrastructure or whether the record is an ELG compatible service;
- the search elements for the "my validation tasks" page (for validators) are the following: name or title, publication status of the metadata record, entity type, the curator of the records, the service registration status for the ELG compatible services, the status of the validation (i.e., technical, metadata and legal), whether the record is resubmitted for validation, and whether the record is accompanied by data uploaded at the ELG infrastructure.

Similar to the catalogue index, the results are returned in JSON format for both these cases and they can be sorted alphabetically by item type or by the date of the last update (for the "my items" page) and the date of submission for publication (for the "under validation" records). Furthermore, in addition to the indexed elements, the under validation records return supplementary information: the validators, previous reviewers' comments and validators' notes if they exist. It should be noted that advanced search functionality with the Lucene Query Syntax and synonyms is not supported for the dashboard indices.

#### **5.1.4 Components and Procedure for the Upload/Download and Storage of Content Files**

Resources can be hosted in ELG, i.e., their metadata descriptions can be accompanied with their content (physical) files. The uploaded files are stored in a private S3 storage bucket, as described in section 3.3.

Data uploads are performed through an S3 Storage proxy (see Figure 2). The proxy communicates with the catalogue backend which authorises the upload. In this way, we allow only the curator of a metadata record to upload the files to be attached to this record. Multiple content files may be uploaded for each metadata record (e.g. for resources that are available in different data formats). All files must be associated with at least one distribution (cf. Section 4.1.1).

A similar process, albeit without the use of the S3 proxy, is used for managing the access granted to users when downloading the datasets. More specifically, upon receiving a download request from the catalogue UI, the backend checks the user's access permissions depending on the user role and the status of the metadata record.

If the metadata record is published, the download is authorised only for users that conform to the requirements imposed by the licensing terms of the specific dataset. For example, data with permissive licences can be directly downloaded by users without further interactions. On the other hand, for data licensed upon condition

of explicit consent, the user is shown the licence document and has to agree to the terms; the backend decides whether this request is eligible taking this consent into account: if the user accepts the terms, it returns a temporary pre-signed S3 download URL to the catalogue UI; otherwise, an appropriate "permission denied" response is returned.

If the metadata record is not published, only its curator, the assigned validators and administrators can download the data, taking into account the publication status of the record; thus, the catalogue backend allows the download of the resource directly, without checking any of the requirements imposed by the accompanied licensing terms.

The catalogue currently accepts compressed files in .zip format and stores information about the size, date of upload and a hash digest (MD5) of the object that reflects changes to its contents; an ETag header is used for including the hash digest to the http request/responses). Every uploaded content file, related to a metadata record, is stored with a unique key, consisting of the metadata record's storage object identifier (a UUID assigned to each metadata record upon creation) and a filename.

### 5.1.5 Registration and Publication of ELG-compliant LT Services

The process of registering an LT service to ELG is as follows:

- The LT provider creates a Docker <sup>38</sup>image that contains an LT tool exposed via an ELG-compliant REST service; the Docker image has to be uploaded to a Docker registry (e.g., GitLab, DockerHub). In some cases, two images are required; one that contains the LT tool and one that implements the ELG API and calls the LT tool via its own custom API. For more information see section 5.2.3.
- The provider creates a metadata record and submits it for publication (see Section 4.2).
- The administrators assign the metadata record to a validator; the record is now visible to the assigned validators.
- The LT service is deployed to the k8s cluster by creating a configuration file<sup>39</sup> and uploading it to the respective GitLab repository. The CI/CD pipeline responsible for ELG deployments (Section 3) will automatically deploy the new service. If requested (by the LT provider), before creating the YAML file, a separate dedicated namespace is created for the LT service<sup>40</sup>. A user with the "technical validator" role assigns to the LT service:
  - The k8s REST endpoint that will be used for invoking it.<sup>41</sup>
  - An id for the service that will be used to call the LT Service via the respective REST API (Section 5.2.1).
  - An appropriate try-out UI for the service type.
- The validator can use the try-out UIs or the command line to test the service; integration issues are identified and solved in collaboration with the LT provider; as a communication medium we use

---

<sup>38</sup> Kubernetes is also compatible with other containers besides Docker; it can host any other Open Container Initiative (OCI)-compliant container (e.g., Kata, <https://katacontainers.io/>). OCI (<https://opencontainers.org/>) is a community project maintained by the Linux Foundation that aims to establish common standards for containers. The OCI format was based on Docker. Currently, Docker is the most popular containerization technology.

<sup>39</sup> <https://gitlab.com/european-language-grid/platform/infra> - this repository is accessible only to the platform administrators

<sup>40</sup> A separate namespace is usually allocated for a set of services.

<sup>41</sup> The endpoint follows this template `http://{k8s service name for the registered LT service}.{k8s namespace for the registered LT service}.svc.cluster.local{the path where the REST service is running at}`.

dedicated Slack<sup>42</sup> channels. This iterative process is continued until the tool/service is correctly integrated. This procedure requires access to the k8s cluster for the validator.

- When the LT service works as expected, the validators approve the metadata record and it is finally published at the public catalogue and, thus, available to all ELG users.

### 5.1.6 Catalogue Population

The catalogue population is performed in one of the following ways:

- registration of metadata records, optionally with accompanying data in the case of LRTs, and in an ELG-compliant form for services, by platform users,
- automatic harvesting from other repositories/catalogues, and
- targeted collection initiatives and surveys aiming to gather as much information as possible for specific categories (for organizations and LRTs so far).

#### 5.1.6.1 Registration of Language Resources and Technologies by ELG Platform Users

ELG platform users can register their language resources via a) the ELG interactive editor, b) by uploading ELG compliant metadata records in single or batch mode, or c) through a functionality that creates metadata records from existing ones. In all cases, the metadata records must adhere to the ELG minimal version.

The **interactive editor** (see also Section 6.1 and Deliverables D3.3 – D3.4) supports users in creating new metadata records, as well as editing/updating existing ones. It includes the mandatory and recommended metadata elements. It supports users in creating "draft" and "syntactically valid" metadata records, as described in Sections 4.2 and 5.1.2.

In the **"upload metadata files"** functionality, users are prompted to upload a file with the description of their resource in XML format. In order to facilitate the registration process, pre-filled metadata templates are available at the ELG GitLab repository<sup>43</sup>.

Finally, a **"copy" functionality** is at the service of users who want to use existing metadata records as a template for generating similar ones (e.g., to create multiple Machine Translation services for different pairs of languages); for new records, users are prompted to add a new name. A similar functionality can be used for creating new versions of a record, in which case users only add the new version number and links to the previous version(s) are automatically created. In both cases, curators can use the interactive editor for further changes.

#### 5.1.6.2 Harvesting Other Catalogues/Repositories

As an additional channel for metadata population, the ELG platform aggregates metadata records from other catalogues and repositories<sup>44</sup>.

The ELG platform implements an OAI-PMH client for harvesting metadata from other repositories which expose their metadata via an ELG-compatible OAI-PMH endpoint. The process of harvesting first requires the registration of a third-party provider as an "OAI-PMH Provider" into the ELG catalogue by an ELG administrator. As soon as communication is established, the third-party provider shares their OAI-PMH endpoint, which ELG will call at regular intervals (currently once a week) in order to harvest their exposed metadata. The ELG harvester

---

<sup>42</sup> <https://slack.com>

<sup>43</sup> <https://gitlab.com/european-language-grid/platform/ELG-SHARE-schema/-/tree/master>

<sup>44</sup> The import task is performed both manually and automatically. The procedure of manual identification and import of records into the ELG platform is described in Deliverables D5.1 – D5.3. In this deliverable, we present the automatic procedures and functionalities.

accepts metadata records compliant with the ELG metadata schema minimal version; the ELG team supports the providers in the process of conversion of their original schema into the ELG one. Currently, we accept OAI-PMH harvested metadata from ELRC-SHARE<sup>45</sup> and three national CLARIN consortia, namely LINDAT/CLARIAH-CZ<sup>46</sup>, CLARIN-PL<sup>47</sup> and CLARIN-SI<sup>48</sup> (see D5.3 for more details). The harvested metadata get the "published" status and are immediately visible at the ELG public catalogue.

A similar procedure is under implementation for catalogues that expose metadata with the OAI-PMH protocol, yet in other schemas. The difference in this case is that the conversion of the metadata records into the ELG schema is performed at the ELG side. For this reason, we are in the process of mapping the DCAT vocabulary into the ELG schema, and implementing the relevant converter; DCAT was selected due to its widespread use among catalogues with datasets. This approach will be used for the harvesting of metadata records from Zenodo.

Zenodo presents an additional challenge for the harvesting procedure: it is a general catalogue for all types of research outcomes, e.g., publications, datasets, posters, images, etc., and for a wide range of research communities. Since for ELG purposes, we are only interested in datasets and software that are of use to the Natural Language Processing (NLP) and LT communities, we are experimenting with high-precision filtering methods that will allow us to identify records of interest out of more than 2,000,000 records, of which around 600,000 for datasets and software.

A different procedure is used for catalogues that expose metadata records through custom APIs and proprietary metadata schemas. This procedure is used only for catalogues that are of high interest to the ELG objectives, following agreements, and has already been used for the import of records from Hugging Face<sup>49</sup>.

The catalogue of Hugging Face includes a large collection of datasets and ML (Machine Learning) models with a focus on transformers. Although Hugging Face encourages adding descriptions for the resources, this is not strictly enforced, and the suggested metadata elements do not follow a standard schema. Hugging Face exposes two distinct APIs with JSON files for datasets and ML models respectively. These JSON files include a subset of the metadata elements displayed on their catalogue and do not cover the minimal version of the ELG schema<sup>50</sup>. Thus, the conversion and import of records from Hugging Face into ELG was restricted to datasets and only to those that have filled in at least a description, and values for the language and licence elements, which are deemed the minimum threshold for the findability and usability purposes in the context of ELG. A custom converter was developed based on the mapping of the elements and, in the case of controlled vocabularies, their values. Manual work was further necessitated for the enrichment of information for specific elements<sup>51</sup>.

#### 5.1.6.3 ELG Collection and Crowdsourcing Initiatives

In R2, we have initiated another form for populating the ELG catalogue with bulk lists of metadata records (potentially with limited information) that serve as seed for further enrichment.

---

<sup>45</sup> <https://www.elrc-share.eu>

<sup>46</sup> <https://lindat.mff.cuni.cz>

<sup>47</sup> <https://clarin-pl.eu/dspace/>

<sup>48</sup> <https://www.clarin.si>

<sup>49</sup> <https://huggingface.co>

<sup>50</sup> At the time of the import from Hugging Face, the relaxed version of the schema was not considered. At following stages, this will be used to increase the number of imported records.

<sup>51</sup> For more information, see Deliverable D5.3.

This mode has been used for the population of the catalogue with organisations that use or develop LT applications, thus enabling us to quickly create "yellow pages" for organisations active in the broader LT area. For this purpose, we have merged lists of organisations from various sources, together with information on them – mainly contact data and key terms that describe their LT-related activities. This list, divided into sub-lists by country, was checked by the respective National Competence Centres (NCCs)<sup>52</sup>. The revised list, with more than 1,700 records, was uploaded into the ELG catalogue in two phases, first for companies and public organisations and at a later stage for academic organisations.

The records that have been added in this way can be "claimed" by interested individuals that work for these organisations to be further enriched (Figure 9). When a person claims a metadata record, the ELG administrators are notified and can approve or reject the claim, taking into account the professional email account of the user; if the claim is approved, the metadata record is unpublished and assigned to the user for further editing. Once the user finishes the editing, the record is submitted for publication and goes through the normal publication procedure.

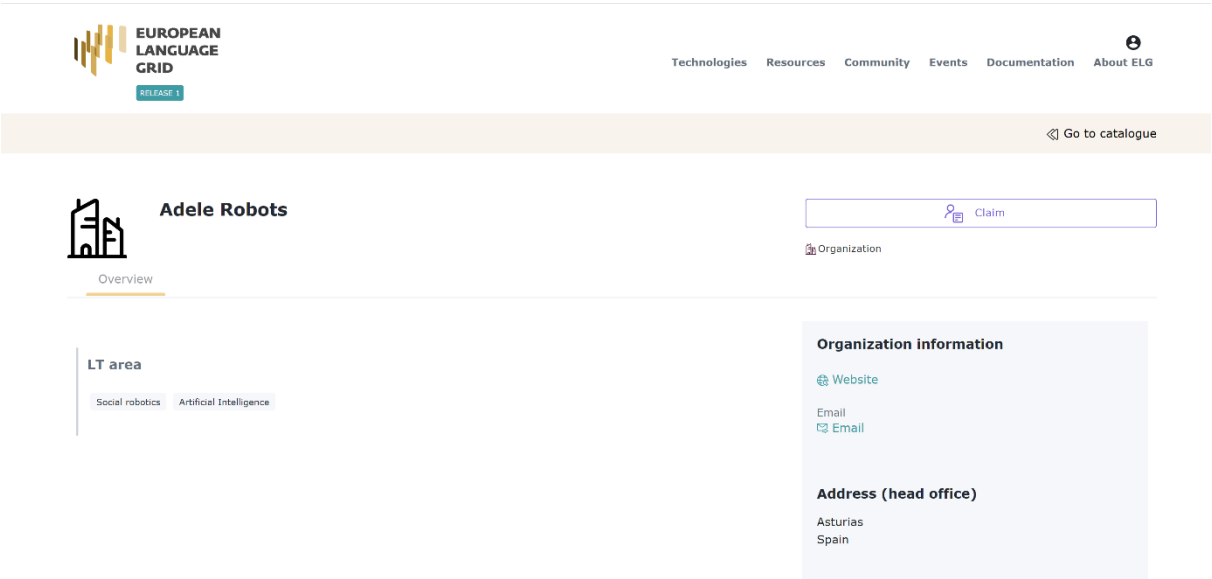


Figure 9: Claim of metadata records

During the timeframe of R3, the European Language Equality (ELE) project<sup>53</sup>, which collaborates with ELG to promote digital language equality in Europe, has launched a survey<sup>54</sup> to collect information on LRTs available for the languages under investigation. The collection of information was performed through a web form designed for this purpose that included a subset of the ELG schema mandatory metadata elements. Given the size of the survey (over 26 informants throughout Europe for over 80 languages, official, regional, minority) and the time restrictions, the demand for filling in even the minimal version was considered unrealistic.

<sup>52</sup> For more information, see Deliverable D7.2.  
<sup>53</sup> <https://european-language-equality.eu/>  
<sup>54</sup> <https://european-language-equality.eu/languages/>

The survey outputs, in the form of spreadsheets, have been curated through semi-automatic and manual procedures by ELG staff and imported into the platform through scripts designed for this purpose. The metadata records (over 6,000) are marked as claimable for further enrichment.

#### 5.1.6.4 Current Contents

At the time of writing, the ELG catalogue includes the following set of public metadata records:

- 2,728 tools/services, of which 503 are fully integrated services provided by the ELG consortium partners, pilot projects and other third parties (see D4.3 for a detailed report)
- 8,875 data resources, consisting in 6,237 corpora, 2,228 lexical/conceptual resources, 360 models, 46 computational grammars and 4 uncategorized language descriptions (see D5.3 for more information);
- 1,788 organisations and
- 34 projects, including the pilot projects (see D6.3).

These numbers are subject to change upwards in the immediate future as a result of the harvesting processes currently going on.

#### 5.1.7 Assignment of Persistent Identifiers

In line with the FAIR principles, and more specifically, *F1 – "(Meta)data are assigned a globally unique and persistent identifier"*<sup>55</sup>, the ELG platform will soon support the assignment of persistent identifiers (PID) for LRTs.

For this operation, we have selected to use the DataCite<sup>56</sup> service as a PID provider that assigns Digital Object Identifiers (DOIs)<sup>57</sup>. DataCite is a leading global non-profit organisation that provides persistent identifiers (DOIs) for research data and other research outputs. Organisations in the research community can join DataCite as members to be able to assign DOIs to all their research outputs, thus enhancing their discoverability. DataCite offers additional services for the DOI management, and facilitates connection and sharing of DOIs in the broader research ecosystem. At the time of the writing, we have initiated discussions on the most appropriate plan for joining DataCite. We have also created a test account that allows us to make simulation of the DOI assignment procedure.

Following the DataCite best practices<sup>58</sup>, we have opted to assign new identifiers only for the ELG-compatible services and ELG-hosted resources. Metadata records (e.g., harvested from other catalogues, or registered by individuals without any content files) will be identified by the URL of their landing page. It should be noted that for resources which already have other identifiers (e.g., handle.net PIDs, DOIs, etc.), the related identifiers are already included in their respective metadata record. The DOI will be minted at the same time the metadata record gets published.

Moreover, we have decided to adopt the DOI versioning scheme introduced by Zenodo<sup>59</sup>. This means that for each resource, one DOI will be registered for each of its versions, and one more, the "Concept DOI", representing all its versions and linking them together. The Concept DOI will always resolve to the landing page of the latest versioned record of the resource, while all other DOIs will resolve to the specific version for which they were assigned. Therefore, users will be able to use in their citations either the DOI for the specific version of a

---

<sup>55</sup> <https://www.go-fair.org/fair-principles/>

<sup>56</sup> <https://datacite.org/>

<sup>57</sup> <https://www.doi.org/>

<sup>58</sup> [https://datacite.org/documents/DataCite\\_BestPractices\\_ServiceProviders\\_v1.pdf](https://datacite.org/documents/DataCite_BestPractices_ServiceProviders_v1.pdf)

<sup>59</sup> Nowak, Krzysztof, Ioannidis, Alexandros, Bigarella, Chiara, & Nielsen, Lars Holm. (2018). DOI Versioning Done Right. Open Repositories 2018 (OR2018), Bozeman, US. Zenodo. <https://doi.org/10.5281/zenodo.1256592>

resource, when they want to refer to the exact artifact used, for instance, in their research work for reproducibility reasons, or the Concept DOI, when they want to refer to the resource without further specifying the version.

The ELG metadata schema (version 3.0.1) has been mapped to the DataCite Metadata Schema (version 4.3). The mapping covers the full DataCite schema, i.e., it includes the mandatory elements as well as the recommended and optional ones, wherever this was feasible. For creating DOIs we are going to use the DataCite REST API and the metadata will be sent in the DataCite JSON format. We are currently in the process of testing this procedure in order to be ready to fully integrate it in the publication lifecycle as soon as the administrative procedures are completed.

### 5.1.8 Export of Metadata Records and Exposure through Other Catalogues

All metadata records in the ELG catalogue are available for download in XML format compliant with the ELG schema.

In addition, we expose the metadata records of LRTs to Google's dedicated search engine for research datasets<sup>60</sup>. To this end, we have mapped a subset of the metadata elements to the schema.org<sup>61</sup> vocabulary and embed in the landing pages of LRTs the generated JSON-LD<sup>62</sup> metadata in the form of structured data<sup>63</sup>.

Following the assignment of DOIs (see Section 5.1.7), we intend to share through the landing pages and REST APIs the metadata records also in the DataCite schema.

## 5.2 LT Processing Services Execution Backend

One of the main goals of ELG is to provide a substantial number of LT services through the platform. To accomplish full integration and deployment of the services, we rely on containerisation and the specification of generic LT processing APIs. For accessing the LT services from the ELG catalogue, from the command line or by using any programming language, a common public REST API has been made available (Section 5.2.1). The LT service execution is powered by Knative, a platform installed on top of K8s that provides scale-down-to-zero and autoscaling (Section 5.2.2).

### 5.2.1 Internal LT APIs

The LT tools we currently work with and integrate into ELG broadly fall into one of the following categories: Information Extraction (IE), Text Classification (TC), Text-to-text generation (most notably Machine Translation (MT), but also summarisation, anonymisation, etc.), Automatic Speech Recognition (ASR), Text to Speech Generation (TTS), plus a small but growing number of services that do not fit the primary categories (e.g., a SPARQL API for accessing linguistic Linked Open Data (LOD)). For each principal category a specific API was defined (Task 2.5) with the aim of standardising the invocation of these services and their integration into the ELG platform. This API specification and the adoption of Docker images for packaging the LT tools solves several interoperability issues and facilitates their deployment in the ELG platform. The ELG requirements for integrating an LT service are the following:

**Expose an ELG compatible endpoint:** An application that exposes an HTTP endpoint for the provided LT tool should be created. The application should consume via the HTTP endpoint requests that follow the ELG API

---

<sup>60</sup> <https://datasetsearch.research.google.com>

<sup>61</sup> <https://schema.org>

<sup>62</sup> <https://json-ld.org>

<sup>63</sup> <https://developers.google.com/search/docs/advanced/structured-data/intro-structured-data>

request format, call the LT tool and produce responses, again in an appropriate ELG JSON format. For instance, for an IE tool, its HTTP endpoint should accept POST requests in the JSON-based format presented in Figure 10.

```
{
  "type": "text",
  "params": { ... }, /* optional */
  "content": "The content, as a string inline",
  // mimeType optional - this is the default if omitted
  "mimeType": "text/plain",
  "features": { /* arbitrary JSON metadata about this content, optional */ },
  "annotations": { /* optional */
    "<annotation type>": [
      {
        "start": number,
        "end": number,
        "features": { /* arbitrary JSON */ }
      }
    ]
  }
}
```

Figure 10: JSON input

Most parts of the request are optional, only "type" and "content" are required. The "features" and "annotations" fields may be useful for services that can build on partially annotated output from other services. The "start" and "end" of each annotation specify the position of the annotation within the text. For services that process audio or image data, the API request is based on the MIME multipart format, with the request metadata in JSON similar to the above but the binary data supplied as a separate MIME part. The JSON response for an IE tool should be in the format presented in Figure 11. The response contains sets of annotations, and, for each annotation, the start and end offsets are included together with any other available features.

```
{
  "response": {
    "type": "annotations",
    "warnings": [ ... ], /* optional */
    "features": { ... }, /* optional */
    "annotations": {
      "<annotation type>": [
        {
          "start": number,
          "end": number,
          "features": { /* arbitrary JSON */ }
        }
      ]
    }
  }
}
```

Figure 11: JSON output

Detailed documentation and examples for all the different cases of requests and responses is available through the ELG documentation<sup>64</sup> and in D4.1.

---

<sup>64</sup> [https://european-language-grid.readthedocs.io/en/stable/all/A3\\_API/LTInternalAPI.html](https://european-language-grid.readthedocs.io/en/stable/all/A3_API/LTInternalAPI.html)



**Containerisation:** The application should be containerised and the respective image should be uploaded into a Docker Registry, such as the GitLab, DockerHub or Azure container registry and made accessible to anyone. Docker images that are private (i.e., access is limited only to its owners) can also be deployed to ELG, by configuring the Kubernetes cluster with the respective credentials/access tokens. Three different options for the provision of LT tools are supported:

- **LT tools packaged in one image:** One Docker image is created that contains the application that exposes the ELG-compatible endpoint.
- **LT tools running remotely outside the ELG infrastructure:** For these tools a proxy image is created that exposes one (or more) ELG-compatible endpoints; the container communicates with the actual LT service that runs externally, i.e., outside the ELG infrastructure.
- **LT tools requiring an adapter:** For tools that already offer an application that exposes a non ELG-compatible endpoint (HTTP-based or other), a second adapter image should be created that exposes an ELG-compatible endpoint and that acts as proxy to the container that hosts the actual LT tool.

### 5.2.2 Kubernetes and Knative

All LT services are packaged as Docker images that follow the ELG specifications. The LT containers as well as the core components of the platform (e.g., the catalogue backend) run in a k8s cluster within predefined namespaces (e.g., for services, the "elg-srv" namespace). For security and isolation reasons, an LT provider can request a dedicated namespace with restricted access for its services. For instance, LT services provided by Expert System run in the "elg-srv-expsys" namespace, with Docker registry credentials configured using the standard k8s "secret" mechanism. When an LT service is deployed using an adapter, the LT service and the adapter containers are deployed in the same k8s pod<sup>65</sup>.

On top of k8s, we installed Knative<sup>66</sup> to efficiently manage the workload of the various LT services and to avoid draining of the available hardware resources. A substantial number of LT services are planned to be integrated into the ELG platform, many of which require a significant amount of hardware resources (CPU and/or memory). It is impossible to keep all of them up and running all the time. Knative addresses this issue by offering a scale-to-zero functionality; i.e., the number of running containers (replicas) of a certain LT service is scaled down to zero for the time period that the service is not used by anyone. Also, Knative offers autoscaling functionalities, i.e., when a user initiates a processing request for a service, knative receives this request and starts the corresponding container (if replicas = 0) and forwards the request to it. Depending on the number of requests for this service and based on the configuration, it spawns additional containers. An example of such a configuration, for an MT service (French to English) created by Charles University, is given in Figure 12.

---

<sup>65</sup> A pod is a group of containers deployed together on the same host. Containers within a pod share a common network stack and can communicate with one another via the local loopback address 127.0.0.1

<sup>66</sup> <https://knative.dev>

```
image: "registry.GitLab.com/european-language-grid/cuni/srv-translation-t2t/fr-en"
noIngress: true
containerport: "8080"

scalability: "dynamic"
minScale: "0"
maxScale: "2"
concurrency: "500"

requests_memory: "2048Mi"
requests_cpu: "5m"
limits_memory: "2560Mi"
limits_cpu: "1000m"
```

Figure 12: Knative configuration example

The "image" field specifies the location of the LT image, "containerport" defines the port where the ELG-compliant REST service hosted in the container runs. The parameters "minScale" and "maxScale" define the minimum and maximum number of replicas that can run for this service. "Concurrency" specifies a target number of concurrent requests to be served by each replica of the LT container; a sustained load above this level will cause additional instances of the container to be spawned, subject to the configured "maxScale" limit. The last four fields specify hardware requirements/limitations for the containers that run for this LT service<sup>67</sup>.

### 5.2.3 LT Service Execution Server and External LT API

The LT Service Execution Server is responsible for orchestrating LT processing. This server is different from the catalogue backend application. It implements all functionalities relevant to LT processing. In this way, we achieve modularity and a clean separation of platform functionalities. The LT Service Execution Server provides:

- a client/connector for sending HTTP processing requests to and receiving HTTP responses from the backend LT services;
- a mechanism for retrieving the k8s REST endpoint of an LT service; this endpoint is configured during the registration of the service and is kept alongside other information in the respective database table;
- a mechanism for limiting access to LT services; i.e., for each registered ELG user<sup>68</sup>, we keep a daily usage record containing the number of requests/calls that have been submitted to the LT service execution server as well as the total size of these requests (in bytes); if the user exceeds the predefined limits/quotas, execution is not allowed anymore;
- a common REST API used for calling an integrated LT service from the catalogue frontend or from the command line;
- simple error handling, i.e., if something goes wrong during processing, an appropriate message is returned to the client that initiated the process request.

Every service is available at a public URL of the same format<sup>69</sup>:

```
https://{domain}/execution/process/{ltServiceID}[?version={version}]
```

---

<sup>67</sup> <https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/>

<sup>68</sup> When registering to the ELG platform, users must read and consent to the ELG Terms of Use, which inform them of their personal data collected when interacting with the platform, and the broader policies of ELG with regard to GDPR (General Data Protection Regulation).

<sup>69</sup> There is a separate convenience endpoint available for services that take "structured text" requests (e.g., text split into paragraphs) to allow users to post plain text and have the execution server split it into segments, rather than having to segment it themselves and build a JSON request.

{Domain} is the domain name of the deployed ELG platform (e.g., the production ELG domain name is "live.european-language-grid.eu"); {ltServiceID} is the id of the service that is being invoked and it is assigned during the registration process and stored in the same database table as the k8s REST endpoint for the specific LT service; the optional {version} parameter is used to select a specific version of the service if there are several to choose from, if omitted then the latest available version is called. The request body may be a complete JSON request as per the ELG specification, or it may be plain text, audio or image data (with the appropriate Content-Type), which will be converted to the relevant API request type by the execution server. In the latter case any other query parameters on the URL (apart from "version") will be forwarded to the LT service as part of the request JSON.

According to a **typical execution scenario**, a user visits the ELG platform, finds a tool (e.g., the ILSP Named Entity Recogniser for English) and goes to the respective resource landing page<sup>70</sup>. The landing page for services includes two tabs that can be used for testing the service with a limited set of calls:

- The "Try out" tab is a specially designed GUI where users can provide a sample input and any parameter values and see the results output by the service. Depending on the service type, they can type in or paste some text, upload or record audio, or select one of the provider's items of sample data if they have supplied any in the service metadata. The results are rendered in a task-specific viewer.
- The "Code samples" tab provides the API endpoint and an example on how to call the service from a user's own code; at the moment, the examples provided are a "curl" command which can be copied and executed locally, and code examples using the ELG Python SDK.

These testing functionalities are reserved for logged in users. In both cases, i.e., when the user clicks on the "Process" button in the "Try out" GUI, or uses the curl command or Python SDK, a POST call, which contains the input data as payload, is sent to the same REST endpoint of the LT Service Execution Server. The LT execution server then communicates with the catalogue backend (via an appropriate endpoint) and checks whether this user is allowed to call this service (based on the respective daily usage/quotas). If access is granted, the catalogue returns the HTTP k8s endpoint of this service. Then, it uses the respective client/connector to generate a request to the endpoint; the request is received from Knative and forwarded to the appropriate container. If there are no available containers up and running for the specified LT service, Knative spawns a new one before forwarding the request (see Section 5.2.2). The response with the output results of the LT service container is returned back to the LT execution server which returns it to the client (try out GUI or curl) that initiated the call.

As described above, only registered users can access the LT service execution server; a first layer of *authentication* is achieved by using oauth2-proxy<sup>71</sup>, an intermediary that is able to validate the JSON Web Token (JWT) credentials issued by the Keycloak Identity and Access Management server. The oauth2-proxy container runs<sup>72</sup> in the ELG cluster, and the nginx ingress controller consults oauth2-proxy when it receives a request destined for the LT execution server. Only requests that contain a valid token are permitted to pass through to the actual LT execution server, where the *authorisation* is performed as described above when the execution server forwards the token to the catalogue to check the user's permissions and quota.

---

<sup>70</sup> <https://live.european-language-grid.eu/catalogue/tool-service/480>

<sup>71</sup> <https://github.com/oauth2-proxy/oauth2-proxy>

<sup>72</sup> A separate Keycloak client was created for oauth2-proxy, see Section 3.2 for more information.

The LT service execution orchestrator is implemented in Java using Spring Boot. Spring Boot was selected because it (a) facilitates application configuration, (b) allows the creation of a consolidated standalone application which can be easily containerised 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 or messaging middleware).

#### 5.2.4 LT Service Helper Services

The ELG platform also includes additional components that are "helper" services available for use by LT service containers. The most important of these is the Temporary Storage service, which provides a mechanism for LT services to return types of output data to the user that cannot be readily represented in the standard JSON response formats. An LT service may POST arbitrary data to the storage service at a known fixed URL resolvable only within the ELG cluster (<http://storage.elg/store>) and will receive in response a *publicly-resolvable* unguessable URL from which the same data can be retrieved for the next few minutes or hours. This URL can be returned to the caller as part of a standard API response. For more detail on this service see deliverable D4.3.

### 5.3 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 billing. These modules are presented in the sections that follow.

#### 5.3.1 User Management

The user management and authentication module is based on Keycloak, an open source identity and access management solution. Its adoption relieves us from the need to design and implement login and registration forms, to create and maintain mechanisms for issuing and distributing access tokens etc.

Many catalogue functionalities do not require authentication (e.g., search, view resource via the catalogue UI); however, some functionalities, such as LT service execution, are provided only for registered users. In these cases, users are redirected to a Keycloak page in order to log in to their account by providing the appropriate credentials. Once logged in, they acquire an access token (in JWT format) and they are redirected back to the catalogue frontend; the token keeps them logged in (for some time). However, a JWT token expires after a time period, the expiration time is configurable and is specified in the Keycloak server; this means that the user must re-authenticate. To avoid this situation we use refresh tokens, which are tokens that are issued to the user by Keycloak and are used by a client (e.g. frontend) to obtain in a way transparent to the user a new JWT access token when the current token expires.

A key concept in Keycloak is the realm, i.e., a dedicated domain that consists of a specific set of users, clients, security policies, access roles etc. Using Keycloak's administration console, we have created a realm dedicated to ELG, and registered the required clients under it. A client is the only way for an application (e.g., the catalogue frontend) to use and access Keycloak. We created a "catalogue frontend" client and defined the roles described above, i.e., those of Consumer, Provider, Metadata validator, Technical validator, Legal validator, Administrator and Content Manager (see Section 4.3).

Currently, as already described in previous sections, granting advanced roles for registered users of the platform is a procedure supervised by the administrators. For instance, to become a provider, a user must request this via the user profile and the request is assessed by an ELG administrator.

User authentication is managed by Keycloak, while user authorisation and assignment of the appropriate access level to a resource or functionality is controlled by a specific set of modules implemented as part of the catalogue backend. These catalogue modules must know which roles are assigned to each user. Given that roles are managed in Keycloak and may change at any time, an interaction layer between the two systems is required. For example, as described above, a user applies for the provider role; the application is processed by one of the administrators; if the request is approved, the administrator uses Keycloak's admin pages to assign the role to the user in question. This change of user roles must be communicated from the Keycloak system to the catalogue backend user module at real time, so that the user is granted the appropriate rights when she logs in to the system. In addition, for specific tasks, the catalogue backend requires access to user information kept in Keycloak, which may also change at any time (e.g., when a user's profile is updated). A typical use case is that of sending notification emails to users, e.g., when a metadata record is submitted for publication, which involves accessing the user's name, surname and email. To solve the syncing issue we have developed a Keycloak plugin that notifies the catalogue backend via an appropriate REST service for any addition or change<sup>73</sup>.

### 5.3.2 Monitoring and Analytics

In this subsection we present the modules used for observing the performance of LT processing services to ensure they function properly, and for recording the usage of LRTs included in the ELG platform.

#### 5.3.2.1 Monitoring of the LT processing services

Monitoring of the LT services consists of testing the ELG services on a regular basis, collecting the results of the tests, and, finally, displaying everything on a monitoring interface. These three different steps correspond to three separate services running on separate pods inside the Kubernetes cluster.

The service testing is carried out by a server created with Flask<sup>74</sup> (a well-known web-application framework) which calls every service using the Python SDK (Section 6.2). It collects three metrics (success or failure, response or error message, request time) for each service call and pushes them to the Prometheus<sup>75</sup> server. Prometheus is the service that gathers metrics (from the services monitoring but also metrics from the Kubernetes cluster) and stores them in databases. To visualise the metrics being collected, we finally use Grafana<sup>76</sup>.

Grafana is accessible from outside the Kubernetes cluster<sup>77</sup> in order to make monitoring metrics accessible to the ELG team.

Two dashboards are currently available on Grafana. One provides an overview of the status (working or not) of the ELG services, and a second one, that is service-specific, visualises metrics related to this service (Figure 13).

---

<sup>73</sup> <https://gitlab.com/european-language-grid/ilsp/Keycloak-events-notifier>

<sup>74</sup> <https://palletsprojects.com/p/flask/>

<sup>75</sup> <https://prometheus.io>

<sup>76</sup> <https://grafana.com>

<sup>77</sup> <https://live.european-language-grid.eu/monitoring/>

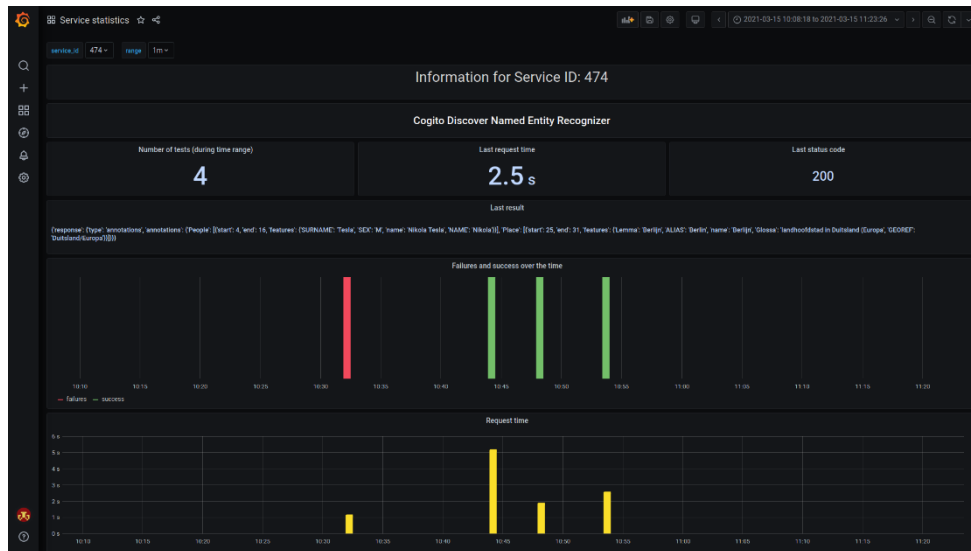


Figure 13: Screenshot of the service-specific Grafana dashboard

### 5.3.2.2 Catalogue usage analytics

The catalogue backend implements a statistics/analytics module for collecting usage information. Currently, the information collected includes the overall number of views for a given metadata record and number of downloads/times used for a resource. User-specific actions are also tracked; these pertain to the downloads and service executions performed by a user.

For user downloads the information collected includes:

- username (if logged in);
- licence attached to the downloaded resource;
- date/time of download;
- relevant metadata record identifier;
- relevant distribution identifier;
- calling client type.

Service execution statistics include

- username;
- service name;
- execution start date/time;
- execution end date/time;
- number of bytes processed;
- execution status (succeeded/failed);
- calling client type.

Statistics tied to individual published metadata records include:

- count of views;
- count of downloads of the resource attached to the record
- count of times used for ELG-compatible services.

It should be noted that in the case of metadata records with multiple versions, the counts for each version are kept separately, but the snippet on the catalogue page accumulates the counts for all versions, given that only the latest version is displayed in the catalogue.

### 5.3.3 Licensing and Billing Module

The ELG platform includes the appropriate mechanisms that support the consumption of resources and services that are available without commercial restrictions. More specifically:

- It supports the download of resources under the condition that they are offered without charges with open access licences or with restrictive licences that require only user authentication and, optionally, formal consent to the licensing terms. Appropriate technical safeguards are implemented to ensure that access to LRTs is granted in accordance with any of the above licensing terms; for instance, access to LRTs distributed with restricted licences is made available only to those users that fulfil the criteria specified in the licences.
- for LT services, only the trial functionality is available and only for registered users (with quotas as described in 5.2).

In R3, we have also worked on a prototype billing module that will enable the ELG platform to offer resources and services distributed with commercial licences. The module should implement a well-defined billing strategy for the ELG platform and for the LRTs it makes available<sup>78</sup>.

The billing module should ensure security and include various services, such as handling subscriptions, payments, pricing, taxes, emails, ensuring customer satisfaction and conformance to all EU and national laws. Furthermore, in order to provide users with a complete solution, it is necessary to offer several functionalities, such as checkout pages, self-service after the payment, cancelation, subscription changes, etc. In order to provide a full and appropriate solution, we have carefully considered various options, starting with developing a full solution from scratch to adopting a commercial platform which meets the needs of the platform and the application scenarios. The evaluation activities concluded with the choice of the existing online commercial platform Chargebee<sup>79</sup>. Deliverable D3.4 provides detailed information about the decision-making process, requirements analysis and a description of the Chargebee functionality.

The choice of a commercial platform brings more attention to the integration architecture and the data flow between the ELG platform and the external billing solution. Figure 14 illustrates the workflow; yellow boxes are components and processes provided by the selected billing platform while the other boxes constitute part of the ELG platform.

---

<sup>78</sup> For a discussion on the strategy, see Deliverable D7.4.

<sup>79</sup> <https://www.chargebee.com>

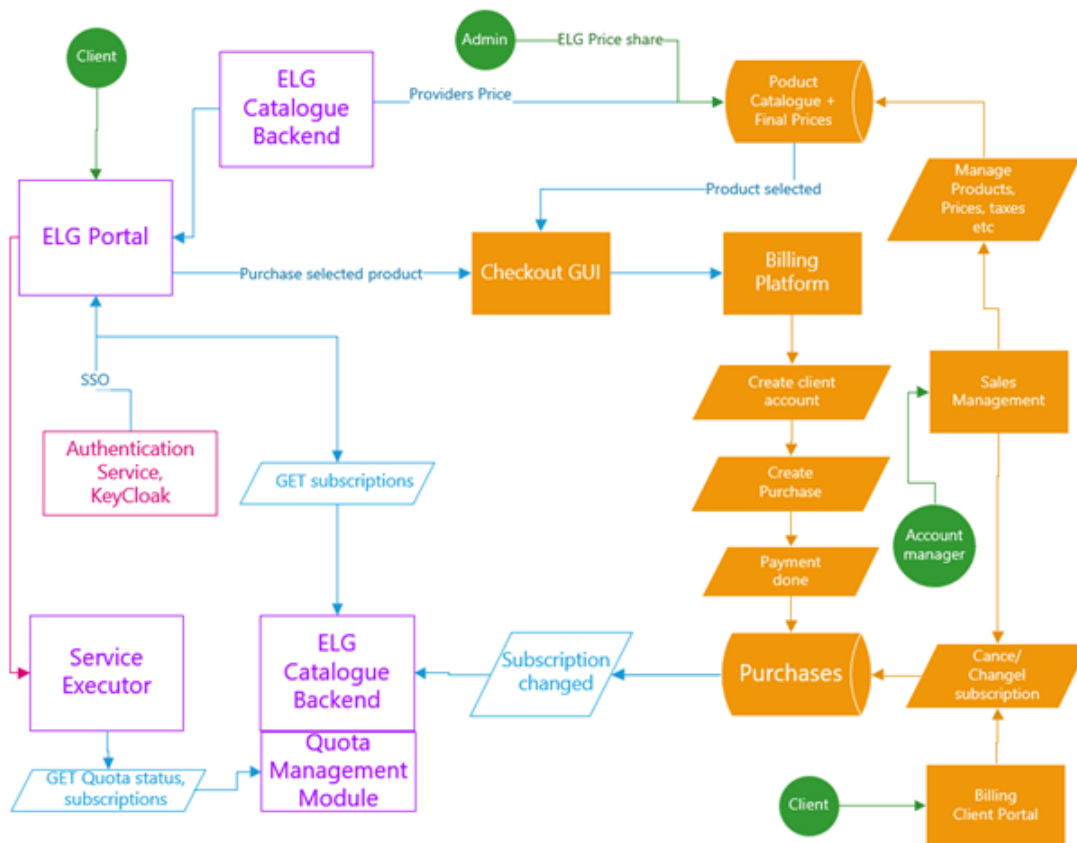


Figure 14: Billing workflow between ELG and Chargebee platforms

The components of the platform that are part of the billing workflow and processes are as follows:

- The **ELG platform** includes ELG-compatible services and hosted resources that are available with different licensing terms; some can be used for free, while others are commercial products, and their providers may wish to offer them with commercial licences. In this case the provider may want some compensation for the usage of their services/resources, in which case the relevant pricing details must be documented for them. For these services/resources, additional execution and administrative costs may be charged by ELG, which are calculated on top of the provider's requested price, and sum up to the final price of the service/resource. Information about the commercial product plans and final prices goes to the online billing solution.
- The **Chargebee catalogue** contains all monetarised products and plans, and their prices. The relationship between the ELG catalogue products and the Chargebee catalogue is not necessarily one-to-one; i.e., Chargebee can contain paid plans that allow the use of multiple products from the ELG catalogue, or the download of multiple resources. The relation between the two catalogues depends on the ELG business strategy.
- The **Chargebee billing platform** is the central online billing point, through which ELG administrators can configure the billing process. The platform includes various integration points with external systems (in our case, the ELG platform) which can be configured. The first integration point is the Checkout pages, which is used to get the purchaser onboarded, and the last one the webhooks, that can inform the ELG platform (i.e., catalogue backend) about the events that took place in the Chargebee platform.
- **Chargebee purchases:** all transactions, subscription changes, logs, billing information, subscription data and similar information are stored at the Chargebee side, i.e., an external database living outside ELG.



Any information needed from Chargebee can be synchronised through the webhook functionalities. For the ELG platform, this information includes: the user (identified by email address) that has performed an action through a subscription plan and/or a purchase, the action performed, the billing plan to which the user subscribed, any actions that were canceled (by the user or because the payment was unsuccessful), any other changes in the subscription information. The webhook sends this information with HTTP Post to the ELG Backend so it can register changes in the ELG system.

- **ELG LT Service Executor:** when the consumer requests to execute a service, the service executor (described in Section 5.2) asks the Catalogue Backend if the user has the right to access this service and, depending on the answer, blocks this request or allows the execution of the LT service.
- **ELG Catalogue Backend/Quota module:** it can determine if the user service call is allowed based on the metadata record of the service and information about the user's subscription plans, synchronised with the external Chargebee platform.
- The **Chargebee Sales Management** (Figure 15) forms part of the billing platform; the account managers, user helpdesk, accountant, business specialists can log in and manage client subscriptions, data, payments, generate reports, export data to be used in other systems, such as the accounting software.

	Subscription Info	Customer Info	Next Renewal ↓	MRR	Created On ↓
<input type="checkbox"/>	ACTIVE ILSP MT for English to modern Greek EUR Monthly 8TUTbM5w56DTHRq	Andis Lagzdins andis@example.com	09 Mar 2022	50.00 € EUR	09 Feb 2022
<input type="checkbox"/>	ACTIVE ILSP MT for English to modern Greek EUR Monthly 199Z5V5K1YL2b	Andis Lagzdins andis.lagzdins@tide.com	07 Mar 2022	50.00 € EUR	07 Sep 2021
<input type="checkbox"/>	ACTIVE ILSP MT for English to modern Greek EUR Monthly 8TM9Q5JUS5V90	Tilde andis.lagzdins@tide.lv	07 Mar 2022	50.00 € EUR	07 Sep 2021
<input type="checkbox"/>	ACTIVE ILSP MT for English to modern Greek EUR Monthly 19AAG20ftrR5Ca5	Tilde andis.lagzdins@tide.lv	03 Mar 2022	50.00 € EUR	03 Aug 2021
<input type="checkbox"/>	CANCELLED ACL RD-TEC EUR Monthly 19AAG5Z7K0R4d	Tilde andish@gmail.com	—	0.00 € EUR	04 Jun 2021
<input type="checkbox"/>	CANCELLED ACL RD-TEC EUR Monthly 19AAG5ZMeA4A03	Tilde andis.lagzdins@tide.lv	—	0.00 € EUR	04 Jun 2021
<input type="checkbox"/>	CANCELLED Tilde MT EN-DE EUR Monthly 199Z7Y5ZMU0VG3H	Tilde andis.lagzdins@tide.lv	—	0.00 € EUR	04 Jun 2021

Figure 15: Chargebee subscription management window

## 6 ELG Platform Access Methods

As described in previous sections, the ELG platform exposes several REST APIs that offer functionalities for (a) browsing and searching the catalogue, (b) creating, updating and retrieving metadata records, (c) executing services, (d) downloading resources, etc. We provide two ways to our users for accessing the ELG platform, a web-based UI and a Python package. Both ways make use of the offered REST APIs, however, the web UI offers access to all functionalities implemented in the backend, while the python package offers only a subset of them (e.g., dashboard and metadata editing functionalities are not offered via SDK).

### 6.1 Access through the Website

The ELG frontend consists of

- the "website" pages, maintained in the Drupal CMS and providing information on the LT domain and activities, the ELG project, etc., and

- the platform UIs that enable users to interact with the platform components, i.e., the ELG catalogue (e.g., search, view metadata, upload), the ELG metadata editor for documenting and registering content, and the user dashboard that supports ELG users in accessing and managing their metadata records and resources.

Deliverables D3.2 – D3.3, which are dedicated to the ELG User Interfaces, provide a detailed account for the ELG platform frontend together with a description of the ELG website and the choices made for implementing it. In order to give an overview of the ELG platform that is as complete as possible, we provide an abridged description of the platform's different interfaces.

### 6.1.1 Catalogue User Interface

The catalogue UI is implemented as a Single Page Application (SPA), which provides fast page updates instead of unnecessary full page reloads. All functionalities are built using React<sup>80</sup>, a JavaScript library for the development of user interfaces; the implementation details are presented in Deliverables D3.2 - D3.4.

The final release of the ELG platform offers the following functionalities via the catalogue UI.

- Browse:** The browse/search page (Figure 15) provides a list of all published entries of the ELG catalogue (e.g., LT tools/services, corpora, models, organisations, etc.) accompanied with a snippet of the description of each entry, a set of tags that have been identified as useful for consumers (e.g., for LRTs, languages, keywords and licences), popularity indicators (number of views and downloads/usage) and specific flags that indicate whether the resource is uploaded at ELG or an ELG-compatible service.

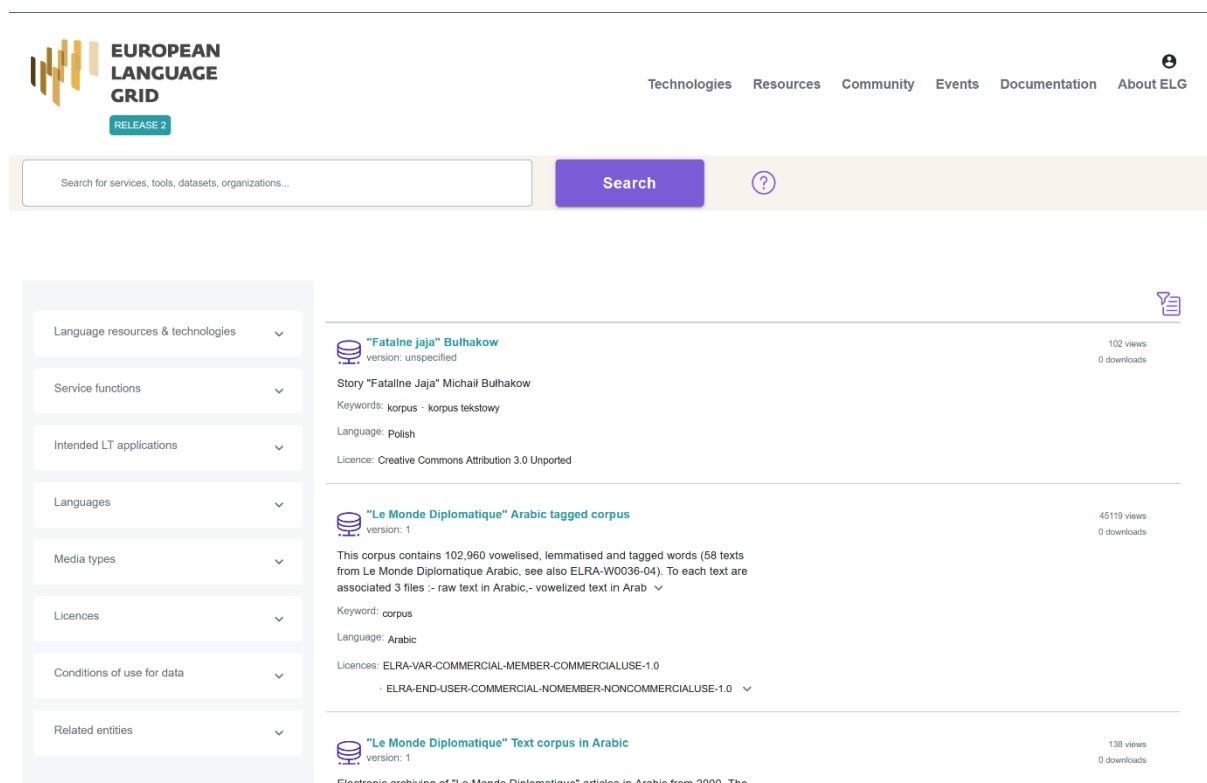


Figure 16: Browse/Search page of the ELG catalogue

<sup>80</sup> <https://reactjs.org>

- **Search:** A user enters a keyword or phrase (Figure 16) in a search box and the application narrows down the results dynamically; the minimum number of characters a user has to type to refine the search results is set to four. The search is performed on the indexed metadata elements (Section 5.1.3).

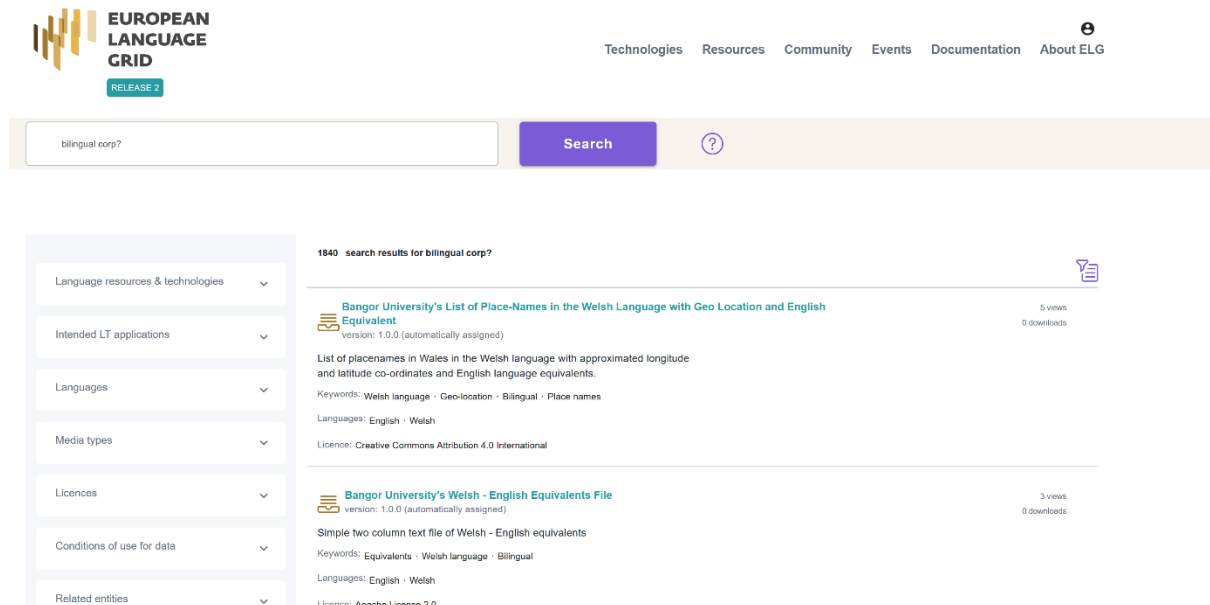


Figure 17: Free text search

- **Faceted search:** Users can filter the full catalogue contents or previous search results by selecting values from the facets on the left side of the browse/search page (Figure 15). The facets have been selected based on user preferences (see D3.1). Facets and free text search can be combined in order to refine search queries and support users in easily finding the resources they desire. Accordions are used to collapse the facets so that they are all visible at once, while facets with long lists of values (e.g., languages, service functions, licences) are endowed with a simple string-based lookup functionality to help users easily find the values they seek for.
- **View published items:** By clicking the title of an item, users can view their full description; Figure 17 displays the landing page of a tool/service. Landing pages have been implemented for all LRT types as well as for Organizations and Projects. The design of the landing pages takes into consideration user preferences (see D3.1), design and accessibility considerations and the ELG metadata schema. As described in more detail in D3.3 – D3.4, the information is grouped in semantically similar sets and organised in tabs and specific areas of the page layout.

**GATE: COVID-19 claim categoriser**  
covid19-misinfo  
Version: 1.0.0  
ELG-compatible service (service running on the provider's side)

Overview Download/Run Try out Code samples

A machine learning classifier trained to categorise claims about COVID-19 into 10 categories proposed by the Reuters Institute for the Study of Journalism.

**Keyword**  
Text categorisation Covid-19  
misinformation  
Information extraction

**Intended application**  
Text categorization  
Information Extraction

**Input content resource**  
Language: English  
Processing resource type: file  
Data format: JSON  
Character encoding: UTF-8  
Media type: text

**Function**  
Function: Text categorization Topic Detection  
Language dependent: yes

**Output resource**  
Language: English  
Processing resource type: file  
Data format: JSON  
Character encoding: UTF-8  
Media type: text  
Annotation type: Certainty level Topic

**Export**  
XML

**All versions**  
GATE: COVID-19 claim categoriser (1.0.0)

**Resource provider**  
GATE Team, University of Sheffield  
Website

**Additional information**  
Landing page

**Resource creator**  
Ian R Roberts  
Email  
Publication date: 17 August 2021

Evaluated: false  
TRL: TRL4

Figure 18: Landing page of an ELG-compatible service

- **Download resources:** All users can download resources directly from the ELG storage system in accordance with their licensing terms. Downloading is offered via the Download tab, which displays all distributions of a resource and information on the licensing (and in the future also billing) conditions for each of them.
- **Try out LT services:** An important benefit for ELG platform users is that they can test the ELG-compliant LT services via pages called "Try out" UIs. This feature is currently reserved for registered users (see Section 5.3.1). The "Try out" UIs are separate HTML pages that use JavaScript code and are embedded as iframes<sup>81</sup> in the catalogue UI. They are responsible for sending and receiving data to and from the LT execution server and for visualising the processing execution results<sup>82</sup>. Figure 18 shows an example. Of course, the "Try out" tab is provided only for those services that are integrated into the ELG platform; this information is provided by the ELG backend. The ELG backend also provides the LT service execution endpoint while the access token that is required is retrieved from the user's active Keycloak session.

<sup>81</sup> [https://www.w3schools.com/tags/tag\\_iframe.asp](https://www.w3schools.com/tags/tag_iframe.asp)

<sup>82</sup> The "Try out" UIs run in containers that are deployed in the same cluster as all the other ELG components.

**Text to Terminological Concept System**  
Text2TCS  
Version: 1.1.2 (15/09/2021)  
ELG-compatible service

Overview Download/Run **Try out** Code samples

Type your own text...  
Type text to annotate

... or select a sample

Beispiel (DE)  
Der COVID-19-Impfstoff von AstraZeneca enthält gentechnisch veränderte Organismen (GVO).

Esempio (IT)  
Il virus denominato SARS-CoV-2 causa la malattia infettiva respiratoria COVID-19.

Questions and answers regarding coronavirus and the COVID-19 disease  
Click to download

Additional parameters

Use?	Name	Value
<input type="checkbox"/>	Skip global relation extraction	false

Figure 19: Try out GUI

- **Validate metadata descriptions:** Users with the "provider" role can validate their metadata records against the ELG schema, before registering them in the ELG platform; appropriate validation errors are provided to the user.
- **Upload metadata descriptions:** Users with the "provider" role can upload metadata descriptions of their resources in XML format compliant with the ELG schema. If the metadata file is valid, the user receives a "success" message; if it's invalid, the validation errors are aggregated and returned in a single message.
- **Describe items with the interactive editor:** For the ELG metadata editor, we have designed and implemented forms that enable users to formally describe their resources in compliance with the ELG metadata schema (Figure 19). Since the schema is rich, we attempt to meet the needs of our users by offering a full-fledged UI that supports easy metadata creation/editing. As a result, we refrain from using overwhelming long forms; instead, we have adopted a more interactive design based on the organisation of the elements along a natural workflow split across multiple steps. The design of the editor takes into account user needs and preferences (see D3.4).

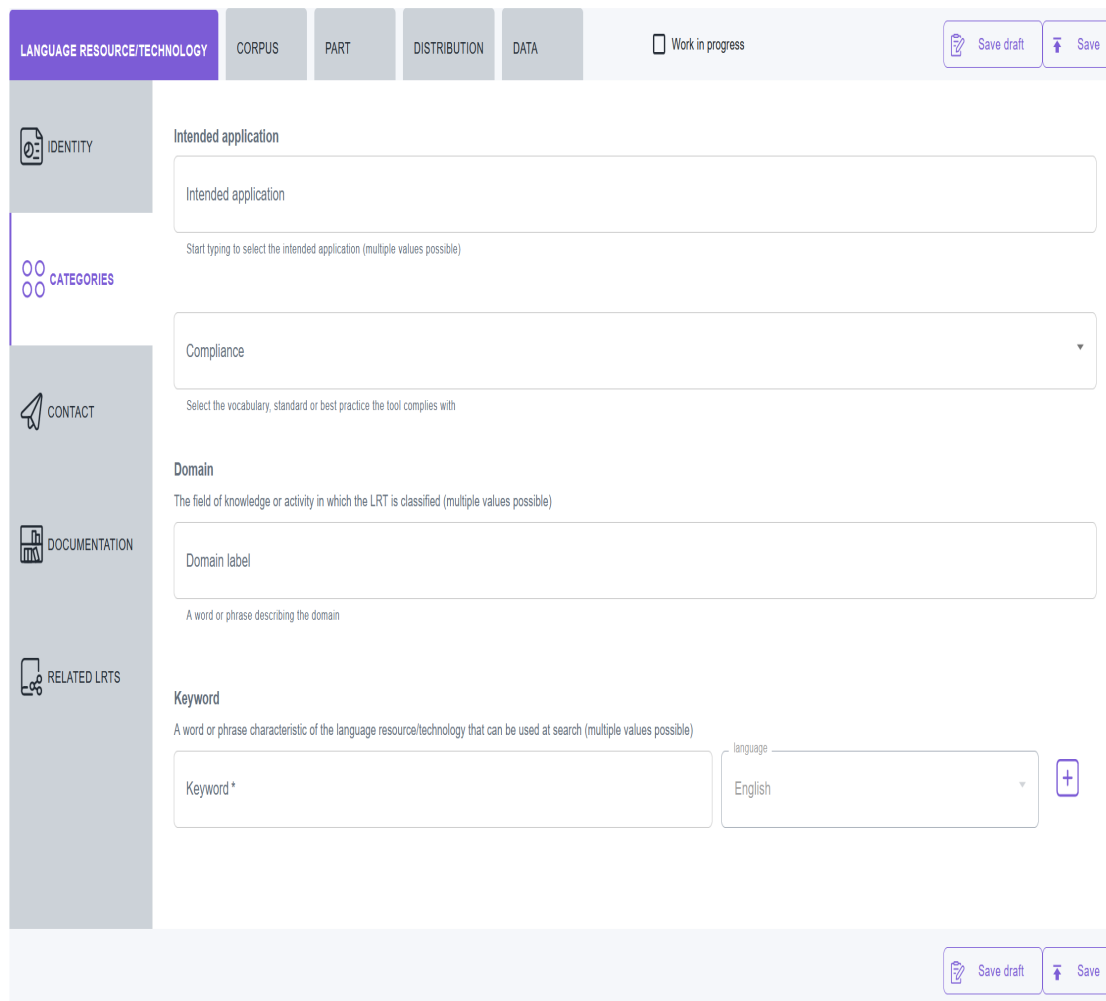


Figure 20: Editor form for corpus

- **Dashboard:** To support users in accessing and managing the catalogue items and performing the actions that are allowed depending on their user role, we make use of a dashboard, which in ELG we call "**grid**" (Figure 20). The grid is the central point through which all users have access to resources and actions, depending on their role. Although the look and feel is the same across user roles, the actions and menu items differ depending on the user role. In this release, we have added functionalities on the provider's grid, and we also implemented the validator's and the consumer's grid.

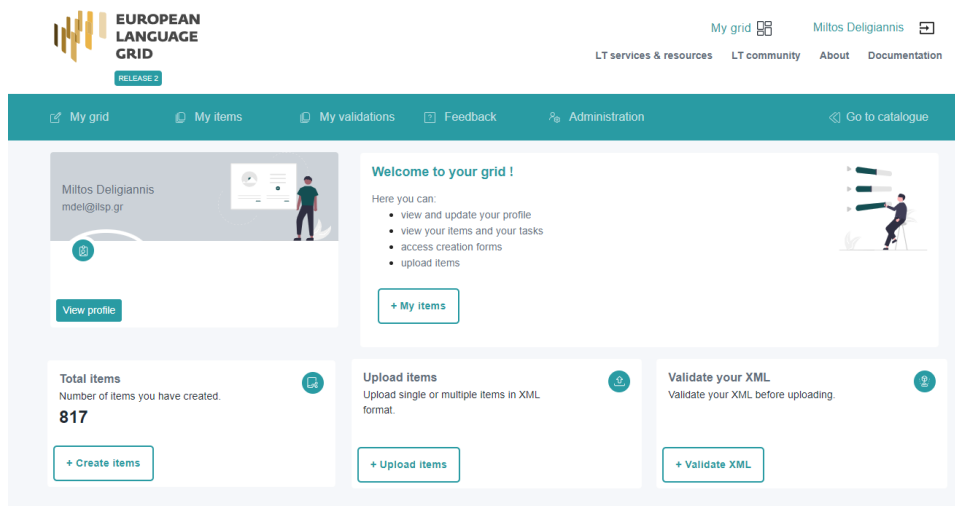


Figure 21: My grid

- Dashboard catalogue pages:** All users access and manage the metadata records through dedicated catalogue pages, which function as a focused version of the catalogue, this time filtering records according to each user's role. These pages implement browse and search functionalities like the main catalogue page. For instance, providers can view the metadata records they have created through the "My items" page (Figure 21) and perform on them the following actions: edit or update a metadata record, submit it for publication, delete a metadata record, upload content files to a metadata record, and replace the content files that have been uploaded.

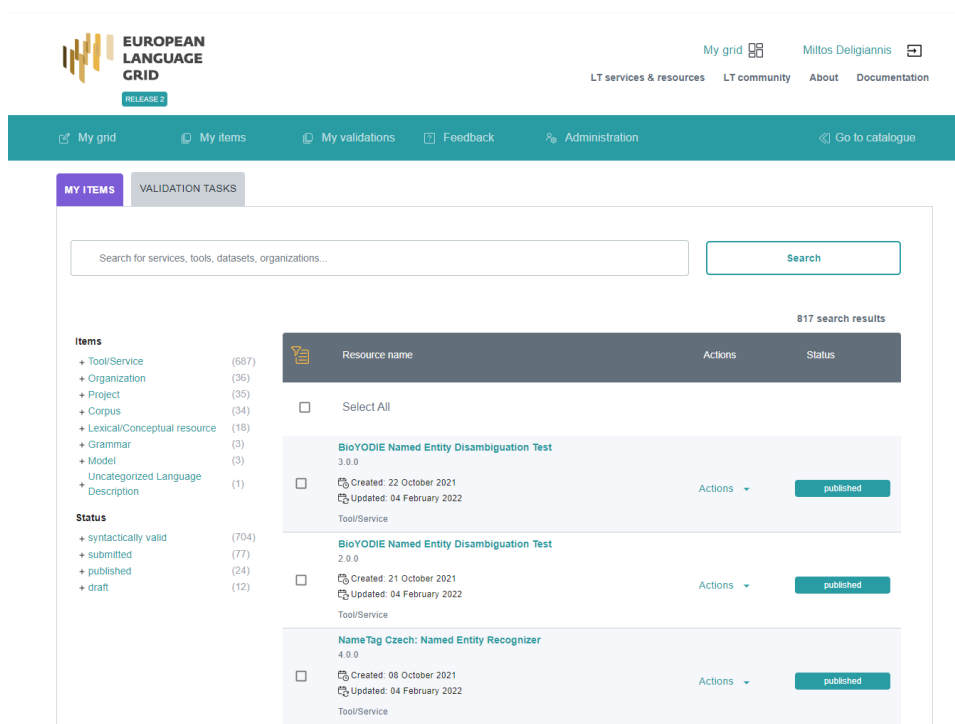


Figure 22: My items

- Validation and service registration pages:** Validators have access to the metadata records that have been assigned to them for metadata, legal and/or technical validation through the "My validation tasks"

page. In addition, technical validators of ELG-compatible LT services have access to the LT service registration forms.

- **Administration pages:** Access to the administration pages, implemented with Django as a separate application, is available only for authorised users (the platform administrators and content managers) via a top-level menu.

### 6.1.2 Integration with the Website

The current entry point for all users of the ELG (production) platform is <https://live.european-language-grid.eu>. This page brings together the website part and the catalogue. Access to the catalogue is available through the "Search" button.

The catalogue UI and the ELG website are implemented as two different applications serving different purposes; however, the integration of the two systems is required for some functionalities. For instance, when a user enters a search query in the website's search field and presses the "Search" button, they are redirected to the catalogue UI where the results of the query are displayed.

In addition, a common "look and feel" is adopted by both applications. The catalogue UI follows the overall design concept of the ELG website and corporate identity to achieve this. Consequently, they both share a similar page layout, with common header and footer sections, and menu items. To integrate these sections, they both consume a REST API provided by the CMS. This API provides the information about which links should be displayed and which links should be enabled and disabled. Finally, both applications make use of common UI frameworks (Google Material design), common icons, fonts and colours.

## 6.2 Python SDK Toolkit

In the framework of ELG, we have developed a Python SDK toolkit to allow users to use ELG through the Python programming language. The result is a pypi package<sup>83</sup> installable using the package installer for Python (pip). The ELG Python SDK provides access to most ELG functionalities through Python. It provides access to the catalogue of resources with methods that allow users to search the catalogue and look for corpora, services, and organisations. The Python SDK enables users to call functional services available in ELG, and even to combine them using a simple pipeline mechanism. In addition to given access to the features of the ELG user interface in Python, the ELG Python package also provides new ones: helper tools to create an ELG compatible service from a Python-based tool and to deploy ELG services locally. We chose Python for the first ELG toolkit as it is the leading coding language for NLP and Data Science projects. The Python SDK's functionalities are detailed below.

### 6.2.1 Browsing the Catalogue

The SDK enables access to the ELG catalogue. It uses the same filters as the UI, i.e., it is possible to filter for the type of resource or LT service, languages, and licence type; free text search can also be used. Figure 22 shows how to search for an English to French MT service. The result is returned as a list of entities where each entity is a Python object that encapsulates the information about the respective ELG resource.

---

<sup>83</sup> <https://pypi.org/project/elg/>



```
from elg import Catalogue
catalogue = Catalogue()
results = catalogue.search(
    resource = "Tool/Service",
    function = "Machine Translation",
    languages = ["en", "fr"],
    limit = 100,
)
print(f"Machine Translation service for English and French:\n{results[0]}")
```

Figure 23: Python SDK Catalogue code example

### 6.2.2 Interaction with the Resources

This section presents the main actions that can be performed with the Python SDK when interacting with resources (downloading ELG hosted resources, creating and running an ELG-compatible service).

#### 6.2.2.1 Obtain an ELG Access Token

In a typical scenario when users attempt to access the ELG platform via the Python SDK, they are redirected to the ELG web interface and asked to enter their login credentials; from this procedure, a JWT token is acquired which keeps them logged-in in order to be able to interact with the platform (see Section 5.3.1). The ELG Python SDK manages authentication based on the Authentication class which communicates with the Keycloak server to obtain Bearer JWT tokens for the specified user. JWT tokens expire after a time period and users have to provide again their credentials. To avoid this, we implemented utility methods that keep users authenticated in a transparent way by using the Keycloak's refresh token mechanism (see Section 5.3.1).

#### 6.2.2.2 Download ELG Corpora

The Python SDK has a Corpus class that corresponds to a corpus or dataset. It can be initialised using the identifier of the resource. If the resource is stored in ELG, it can be downloaded using the download method of the Corpus class. Figure 23 shows how to search for a corpus of German that includes the term "ner" in the indexed elements.

```
from elg import Corpus, Catalogue
catalogue = Catalogue()
results = catalogue.search(
    resource = "Corpus",
    languages = ["German"],
    search="ner",
    limit = 100,
)
corpus = Corpus.from_entity(results[0])
corpus.download()
```

Figure 24: Python SDK Corpus code example

#### 6.2.2.3 Call ELG-compatible Services

The Service class of the Python SDK is very important as it allows users to call ELG services and so to integrate ELG services in Python workflows. It also uses features that simplify the use of services. For example, as mentioned previously, the JWT access tokens are updated automatically (refresh token), the input provided to the service can be either a text or a file, the call can be synchronised or not. The service can be initialised using its ELG id, as shown in Figure 24, or using the result of a catalogue search.

```
from elg import Service
service = Service.from_id(474)
result = service("Nikolas Tesla lives in Berlin.")
print(f"\n{result}")
```

Figure 25: Python SDK Service code example

#### 6.2.2.4 Create an ELG-compatible Service

The Python SDK includes helper tools to create an ELG compatible service from a Python-based tool. First, the package provides two Python classes that can be extended to create a simple HTTP server that exposes an ELG compatible endpoint of the LT tool. Secondly, the ELG Python package comes with a CLI that helps with the creation of the Docker image. The ELG documentation includes a complete tutorial on how to create an ELG compatible service using the ELG Python.<sup>84</sup> With this feature of the Python package, we seek to facilitate as much as possible the creation of an ELG-compatible service from a Python LT tool. Moreover, the services created using the Python helper tools use the best practices to create the endpoint and the Docker image. It makes the services deployed in the ELG infrastructure efficient and secure which is valuable from the ELG point of view.

#### 6.2.2.5 Deploying ELG Services Locally

ELG provides the possibility to call the services deployed in the ELG cluster, using the GUI, or using the API. However, for some cases, it is needed to deploy ELG services locally, e.g., call services without access to internet, make more calls than allowed by ELG, test services not already deployed in the cluster, etc. The Python SDK comes with a command line interface (CLI) that helps to deploy and test ELG-compatible services locally by automatically generating the configuration files to deploy all the needed ELG core components: LT execution server, UI, i18n resolver, temporary storage, in addition to the services themselves. All the ELG core components are published as multi-arch images and therefore can run on different types of processor architecture like x86\_64 (e.g., intel) or arm64 (e.g., M1 macs or AWS Graviton2), however, the LT services only have to run on the x86\_64 architecture (the one used in the cluster) and might therefore not be deployable on all hardware. In addition, to be deployed locally, an LT service needs to be public and have a license that allows it.

The local installation of the services is based on Docker Compose<sup>85</sup>. The Python SDK generates the docker-compose.yml file in addition to all the configuration files automatically, and the users only have to start the applications by running "docker-compose up". The Python SDK covers different local installation setups. It is possible to deploy locally ELG-compatible services deployed in the grid or ELG-compatible services not already deployed in the grid using only the Docker image and the execution location. Also, it is possible to deploy the GUI to interact with the services deployed locally directly from the browser.

To deploy a single service with the GUI, users only have to run the following command:

```
> elg local-installation ids 9192
```

This will generate all the files to deploy the LT execution server, the test UI, the i18n resolver, and the service with the id 9192<sup>86</sup> which is a MT service. Once started using "docker-compose up", the service can be called locally using the API endpoint exposed by the LT execution server or using the GUI (see Figure 25).

---

<sup>84</sup> [https://european-language-grid.readthedocs.io/en/stable/all/A1\\_PythonSDK/TutoServiceIntegration.html](https://european-language-grid.readthedocs.io/en/stable/all/A1_PythonSDK/TutoServiceIntegration.html)

<sup>85</sup> <https://docs.docker.com/compose/>

<sup>86</sup> <https://live.european-language-grid.eu/catalogue/tool-service/9192>

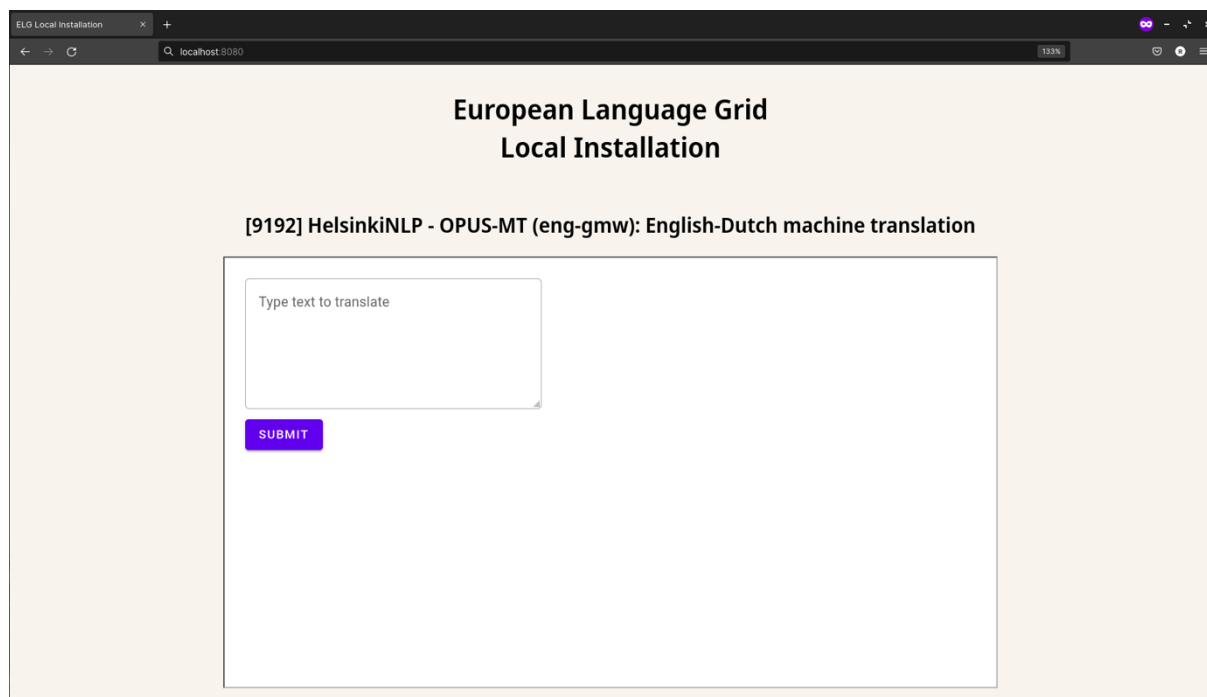


Figure 26: Screenshot of the local installation GUI

## 7 Conclusions

In this deliverable we present the final release of the ELG platform. Users can browse through the catalogue, search for specific LT services and resources, view them, test LT services, and download resources with open licences through graphical UIs or a Python SDK toolkit. Providers can describe and integrate LT services or upload data resources into the platform with an interactive editor, as well as an upload functionality of metadata records. They can also manage their records via their dashboard. Validators have access to their assignments and can perform their tasks through their own dashboard. The APIs required for the execution of LT services, and the interaction with the platform have all been implemented. The publication lifecycle is fully supported and a system for the assignment of persistent identifiers for resources under the control of ELG is soon to be integrated. A billing prototype showcasing the potential of the ELG acting as a marketplace for commercial services and resources has been developed. Bridges to other infrastructures and catalogues have been setup through standard protocols as well as APIs and customised solutions.

## 8 References

Penny Labropoulou, Katerina Gkirtzou, Maria Gavrilidou, Miltos Deligiannis, Dimitris Galanis, Stelios Piperidis, Georg Rehm, Maria Berger, Valérie Mapelli, Michael Rigault, Victoria Arranz, Khalid Choukri, Gerhard Backfried, José Manuel Gómez Pérez, and Andres Garcia-Silva. "Making Metadata Fit for Next Generation Language Technology Platforms: The Metadata Schema of the European Language Grid". In Nicoletta Calzolari, Frédéric Béchét, Philippe Blache, Christopher Cieri, Khalid Choukri, Thierry Declerck, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the 12th Language Resources and Evaluation Conference (LREC 2020)*, Marseille, France, 2020. European Language Resources Association (ELRA).

Georg Rehm, Maria Berger, Ela Elsholz, Stefanie Hegele, Florian Kintzel, Katrin Marheinecke, Stelios Piperidis, Miltos Deligiannis, Dimitris Galanis, Katerina Gkirtzou, Penny Labropoulou, Kalina Bontcheva, David Jones, Ian Roberts, Jan Hajic, Jana Hamrlová, Lukáš Kačena, Khalid Choukri, Victoria Arranz, Andrejs Vasiljevs, Orians Anvari, Andis Lagzdīņš, Jūlija Meļņika, Gerhard Backfried, Erinç Dikici, Miroslav Janosik, Katja Prinz, Christoph Prinz, Severin Stampler, Dorothea Thomas-Aniola, José Manuel Gómez Pérez, Andres Garcia Silva, Christian Berrío, Ulrich Germann, Steve Renals, and Ondrej Klejch. "European Language Grid: An Overview". In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Christopher Cieri, Khalid Choukri, Thierry Declerck, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the 12th Language Resources and Evaluation Conference (LREC 2020)*, Marseille, France, 2020. European Language Resources Association (ELRA).