



# EUROPEAN LANGUAGE GRID

D3.2

Platform GUI  
(initial release)

---

Authors:	Andis Lagzdīņš (TILDE), Uldis Siliņš (TILDE), Jūlija Meļņika (TILDE), Penny Labropoulou (ILSP), Athanasia Kolovou (ILSP), Dimitris Galanis (ILSP)
Dissemination Level:	Public
Date:	31-03-2020

## About this document

Project	ELG – European Language Grid
Grant agreement no.	825627 – Horizon 2020, ICT 2018-2020 – Innovation Action
Coordinator	Dr. Georg Rehm (DFKI)
Start date, duration	01-01-2019, 36 months
Deliverable number	D3.2
Deliverable title	Platform GUI (initial release)
Type	Other
Number of pages	26
Status and version	Final – Version 1.0
Dissemination level	Public
Date of delivery	Contractual: 31-03-2020 – Actual: 31-03-2020
WP number and title	WP3: Grid Platform – Interactive interface and Information System
Task number and title	Task 3.2: Design, development and deployment of ELG front-end Task 3.3: Set up and maintenance of CMS for the ELG
Authors	Andis Lagzdīņš (TILDE), Uldis Siliņš (TILDE), Jūlija Meļņika (TILDE), Penny Labropoulou (ILSP), Athanasia Kolovou (ILSP), Dimitris Gkoumas (ILSP)
Reviewers	Jan Hajič (CUNI), Jana Hamrlová (CUNI), Victoria Arranz (ELDA), Khalid Choukri (ELDA)
Consortium	Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany Institute for Language and Speech Processing (ILSP), Greece University of Sheffield (USFD), United Kingdom Charles University (CUNI), Czech Republic Evaluations and Language Resources Distribution Agency (ELDA), France Tilde SIA (TILDE), Latvia Sail Labs Technology GmbH (SAIL), Austria Expert System Iberia SL (EXPSYS), Spain University of Edinburgh (UEDIN), United Kingdom
EC project officers	Philippe Gelin, Alexandru Ceausu
For copies of reports and other ELG-related information, please contact:	DFKI GmbH European Language Grid (ELG) Alt-Moabit 91c D-10559 Berlin Germany  Dr. Georg Rehm, DFKI GmbH georg.rehm@dfki.de Phone: +49 (0)30 23895-1833 Fax: +49 (0)30 23895-1810  <a href="http://european-language-grid.eu">http://european-language-grid.eu</a> © 2020 ELG Consortium

## Table of Contents

List of Figures	4
List of Acronyms	5
1 Introduction	6
2 Front-end Architecture Overview	6
3 Central Portal	7
3.1 Technical Description	7
3.2 Functionality and Web Interfaces	10
4 Content Management System	11
4.1 Technical Description	11
4.2 Functionality and Web Interfaces	12
5 Catalogue UI	15
5.1 Technical Description	15
5.2 Functionality and Web Interfaces	16
5.2.1 Catalogue Browse and Search Page	17
5.2.2 Landing Page for Items	19
6 Authentication Solution	23
6.1 Technical Description	23
6.2 Functionality and Web Interfaces	24
7 Conclusion	26

## List of Figures

Figure 1. GUI architecture overview _____	6
Figure 2. Angular application life cycle _____	8
Figure 3. Visual Studio Code _____	9
Figure 4. Using the same design and style elements in the different solutions _____	10
Figure 5. New homepage _____	11
Figure 6. CMS life cycle _____	12
Figure 7. Landing page in the CMS _____	13
Figure 8. Header and Footer menus _____	13
Figure 9. Menu link editing _____	13
Figure 10. Editing a menu type _____	14
Figure 11. Menu items received as a JSON object through REST _____	14
Figure 12. Menu permission mechanism _____	15
Figure 13. Performance metrics _____	16
Figure 14. List view of catalogue contents _____	18
Figure 15. Card view of catalogue contents _____	18
Figure 16. Header area of the catalogue page _____	18
Figure 17. Header for an organization with the organization logo _____	19
Figure 18. Header for an organization with default icon _____	19
Figure 19. Main section of a corpus _____	20
Figure 20. Landing page for an Organization _____	20
Figure 21. Landing page for a tool/service _____	21
Figure 22. Landing page for a project _____	22
Figure 23. Bottom content area for resources _____	22
Figure 24. Tabs for functional services _____	23
Figure 25. Test/Try out tab for a sample ASR service _____	23
Figure 26. Login form _____	24
Figure 27. New user registration form _____	25
Figure 28. Password recovery form _____	25
Figure 29. Keycloak user profile form _____	26

## List of Acronyms

API	Application Programming Interface
CMS	Content Management System
CSS	Cascading Style Sheets
ELG	European Language Grid
GPL	GNU General Public Licence
GUI	Graphical User Interface
HTML	Hypertext Markup Language
JWT	JSON Web Token
LT	Language Technologies
MVP	Minimum Viable Product
POS	Part of Speech
REST	Representational State Transfer
SEO	Search Engine Optimisation
SME	Small and Medium Size Enterprises
SPA	Single Page Application
SSR	Server-Side Rendering
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
W3C	World Wide Web Consortium
WCAG	Web Content Accessibility Guidelines
WYSIWYG	What You See Is What You Get

# 1 Introduction

This deliverable outlines the state of development of the ELG platform graphical user interface (GUI). The most recent GUI interface, i.e., the current live system, is available at <https://live.european-language-grid.eu>.

The structure of this document is as follows: after the introduction (Chapter 1), we provide a short overview of the front-end architecture (Chapter 2), which includes all developed solutions and illustrates their interaction through a diagram. Chapters 3, 4, 5, and 6 describe the specific GUI solutions, i.e., the central portal, the CMS, the catalogue UI, and the authentication solution. For each solution we provide a technical description, including implementation details, functionality and web interface descriptions as well as screenshots.

This report has two follow-ups: Deliverable D3.3 (due in M26) and Deliverable D3.4 (due in M34).

# 2 Front-end Architecture Overview

A Graphical User Interface (generally referred to as GUI or front-end) communicates with end-users and clients. It is crucial to provide the best possible solution not only from a usability and information architecture point of view but also from a technological perspective. Search engines and information sharing in social networks are essential for the European Language Grid to attract more attention and visitors, which is why the technical solution must support both aspects in the best possible way.

The consortium includes several distributed teams bringing with them different software development backgrounds. The ELG GUIs are developed by fulfilling various tasks and timelines. This challenge must be taken into consideration when finding a good balance for the development process in terms of providing a unified look and feel and, at the same time, enabling the different teams to use their preferred approaches. With the objective of creating one integrated GUI platform for end-users, search engines and social media agents (content scrapers), a dedicated architecture was devised and implemented. It provides a professional user experience and is suitable for distributed development teams and processes.

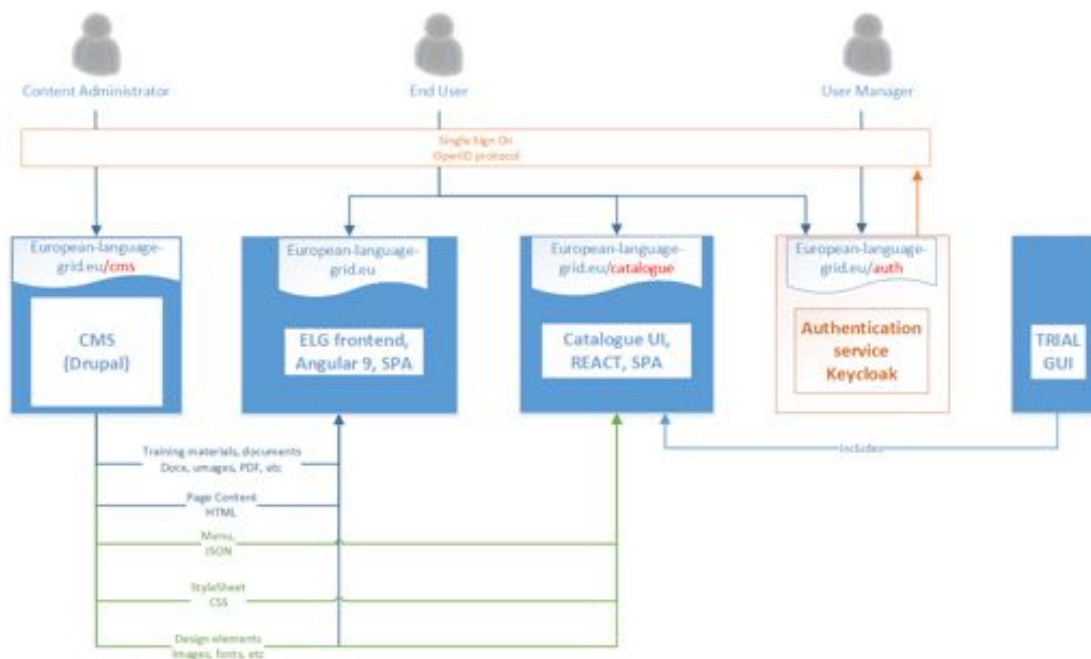


Figure 1. GUI architecture overview

The front-end architecture consists of the following independent software solutions (Figure 1):

1. ELG front-end/Central Portal – the central entry point of the European Language Grid.
2. CMS – provides back-office functionality for static content management.
3. Catalogue UI – for searching, opening, executing LT services, data, projects and other content.
4. Authentication service – an authentication service shared across all GUIs and back-end services.
5. Trial GUIs – multiple small UIs to enable end-users to try out specific LT services.

Both Drupal<sup>1</sup> (CMS) and Keycloak<sup>2</sup> (authentication solution) provide their functionality across different GUI solutions. The CMS shares the design and content information that is reused in different GUI parts, while the authentication solution provides centralized user management and a single sign-on capability to ensure smooth switching between different GUI platforms.

Figure 1 also shows that the CMS shares different assets to all GUIs:

1. Training materials – this section will mainly be dedicated to the central ELG front-end. Users with the appropriate rights are able to upload and manage training materials in various formats, e.g., HTML and PDF files as well as images. These materials will be published by the central portal, but if needed can be also published using the catalogue UI.
2. Page content as HTML – these HTML documents or pages will also be published on the central ELG portal, e.g., content for the entry page, contacts etc.
3. Menus – all menus are stored and managed in a central manner, every GUI can access them and download them in JSON format using REST API.
4. Stylesheets – the CMS is used as a central point for storing the ELG designs used in various parts of the GUI, also in LT service-specific try out UIs. These stylesheets are provided as CSS files.
5. Design elements – the CMS is also used to store design elements such as images, fonts and logos.

## 3 Central Portal

### 3.1 Technical Description

The central entry point of the ELG is served by a single page application (SPA), which is built using Angular<sup>3</sup>. This software retrieves content from the CMS, redirects the user to the catalogue, and allows the user to login.

Figure 2 describes how the Angular application development process is organised. The Angular front-end developers upload their code repository to the cloud platform Azure DevOps<sup>4</sup>. After that, an Azure Pipeline<sup>5</sup> is automatically triggered that pulls the source code for the Angular application, adds configuration values, builds, tests, and tags Docker<sup>6</sup> images, and finally pushes newly created images to the private Azure Container Registry<sup>7</sup>. Each Docker image version is tagged automatically with an auto-increasing build number and with a static tag to control release versions manually.

---

<sup>1</sup> <https://www.drupal.org>

<sup>2</sup> <https://www.keycloak.org>

<sup>3</sup> <https://angular.io>

<sup>4</sup> <https://azure.microsoft.com/en-us/services/devops>

<sup>5</sup> <https://azure.microsoft.com/en-us/services/devops/pipelines/>

<sup>6</sup> <https://opensource.com/resources/what-docker>

<sup>7</sup> <https://azure.microsoft.com/en-us/services/container-registry>

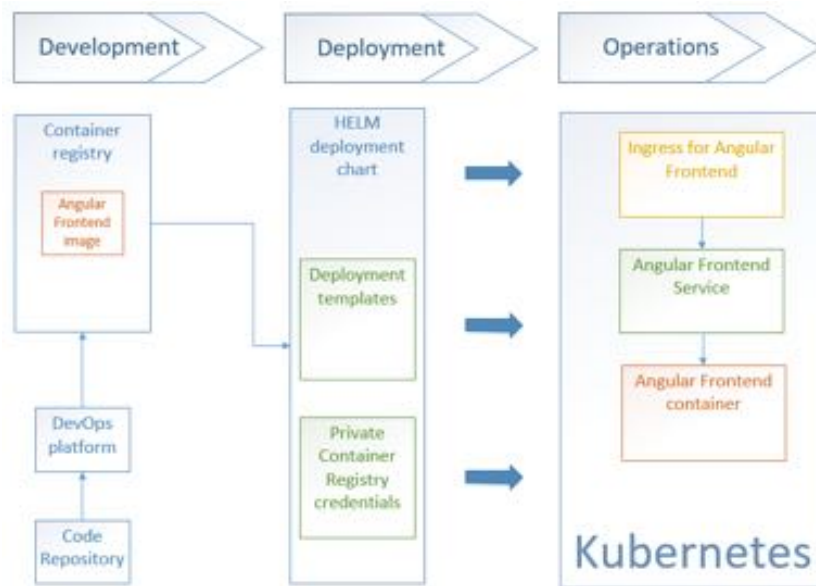


Figure 2. Angular application life cycle

We started the development of the project front-end part with the latest Angular 7.2 version. Since then it has been upgraded twice. We currently work with Angular 9.0. This allows us to keep all the front-end technology up to date, build a long-lasting product and keep the technological stack open to new feature development and fail-proof to updates in other parts of the projects.

One of the challenges of the modern single page application (SPA) built using JavaScript technologies is accessibility to search engine crawlers, which can only access static HTML files. They do not run on the browser, and thus cannot access web pages generated dynamically in the browser when the page is accessed. In these cases, most of the important information from a page is not read. However, Angular can handle this issue. There is a way to develop and deploy an Angular application so that it can provide whole HTML files to search engine crawlers. This approach is called Angular Universal.<sup>8</sup> An application built this way also runs in the browser. The technology represents a good approach to meet all SEO requirements, as well as usability best practices. It works as follows: whenever a user or crawler requests a web page, instead of doing it the usual way – answering user requests with HTML and all JavaScript files – an intermediate step is introduced that renders the response and sends complete HTML files to the user. As soon as this HTML is delivered, the user can see the result immediately, and when the JavaScript is delivered, the application becomes interactive.

Angular is a JavaScript framework built using TypeScript.<sup>9</sup> It is a strongly typed language that gets transpiled to JavaScript. The smart type system of TypeScript helps in the development of safe code.

Visual Studio Code<sup>10</sup> (Figure 3) is used because it is one of the best code editors in the market. It can be extended by various tools written by developers for other developers. It features Angular extensions that help write better TypeScript and detect errors.

<sup>8</sup> <https://angular.io/guide/universal>

<sup>9</sup> <https://www.typescriptlang.org>

<sup>10</sup> <https://code.visualstudio.com>



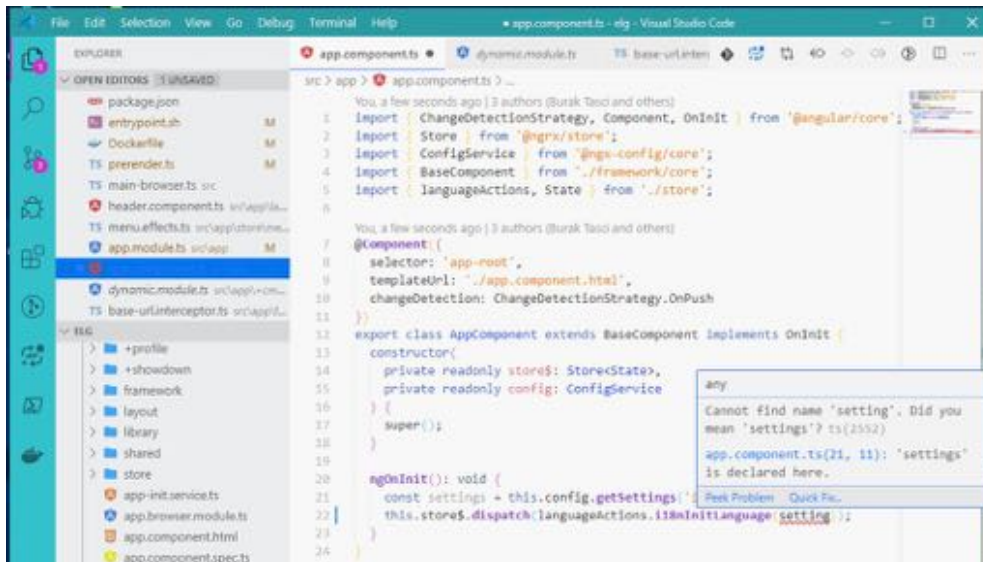


Figure 3. Visual Studio Code

The code is being checked at development time and is inspected for both potentially ambiguous expressions and code that could lead to errors. Visual Studio Code detects many errors before they have a chance to arise.

During a typical work day, a developer writes code, runs scripts, builds it into an application and then runs it in the browser. Developers often do this tens and sometimes hundreds of times per day. It would be smart to help them save development time by building only the parts of the application that have changed since the previous build. Webpack<sup>11</sup> is a tool that bundles all assets into production code. Every time a developer implements a change, the application is rebuilt with new changes taken into consideration. Even if the current ELG is not an extremely large project at this point, completely rebuilding it after every change to see the preview in the browser would be time-consuming and inconvenient. One of the key features of Webpack is Hot Module Replacement. It allows for modules to be replaced without the need for a full browser refresh. It saves valuable time by only updating what has changed and code changes reflect almost instantly.

There is a need for a uniform web design implementation across the whole portal. Recreating every design element from scratch would be unnecessary as there are design systems available that operate with pre-created design element libraries that can be adjusted to the needs of the project. The Google Material<sup>12</sup> design theme was chosen for ELG and it has been useful for different parts of the project. As the application grows and is put together with multiple tools and project parts, there is also a need for a uniform design. Sharing Material design with the CMS, service GUIs and the catalogue has helped to ensure a seamless look on the application's visual parts (Figure 4).

Just before the code is sent from the development machine to the repository, it is automatically tested and built to ensure that only working code is checked in. This is done with the help of the Husky hooks that intercept Git workflows and run code inspections.

<sup>11</sup> <https://webpack.js.org>

<sup>12</sup> <https://material.io/design>



Figure 4. Using the same design and style elements in the different solutions

### 3.2 Functionality and Web Interfaces

The following pages have been created for the central portal:

- *Homepage*
- *Events page*
- *About page* with the following subpages:
  - *Grid*
  - *Consortium*
  - *Open Calls*
  - *Stakeholders*
  - *National Competence Centres*
- *Contact Us page*
- *Terms of Use page*
- Three silo pages are planned and will be activated in the following releases:  
*Technologies, Resources, Community*

The content of these pages mirrors the corresponding pages of the project website (currently available at <https://www.european-language-grid.eu>) to provide a smooth transition for the users.

For ELG Release 1 alpha, the logo has been adjusted to reflect the state of the alpha release and an explanatory text has been added to the homepage. This information will be removed later (Figure 5).

All pages have been created according to the guidelines established in deliverable D3.1. “Requirements and design guidelines”. The design guidelines were updated on 9 January 2020, following a meeting of the design team, which took place on 11 November 2019. The following points have been revised:

- Basic colours (slightly adjusted and later updated to meet WCAG 2 level AA accessibility standard).
- Icon library defined.

- Link to CSS file updated.
- Specific design elements such as colours, elements, icons, text styles updated.
- Several example layouts presented.

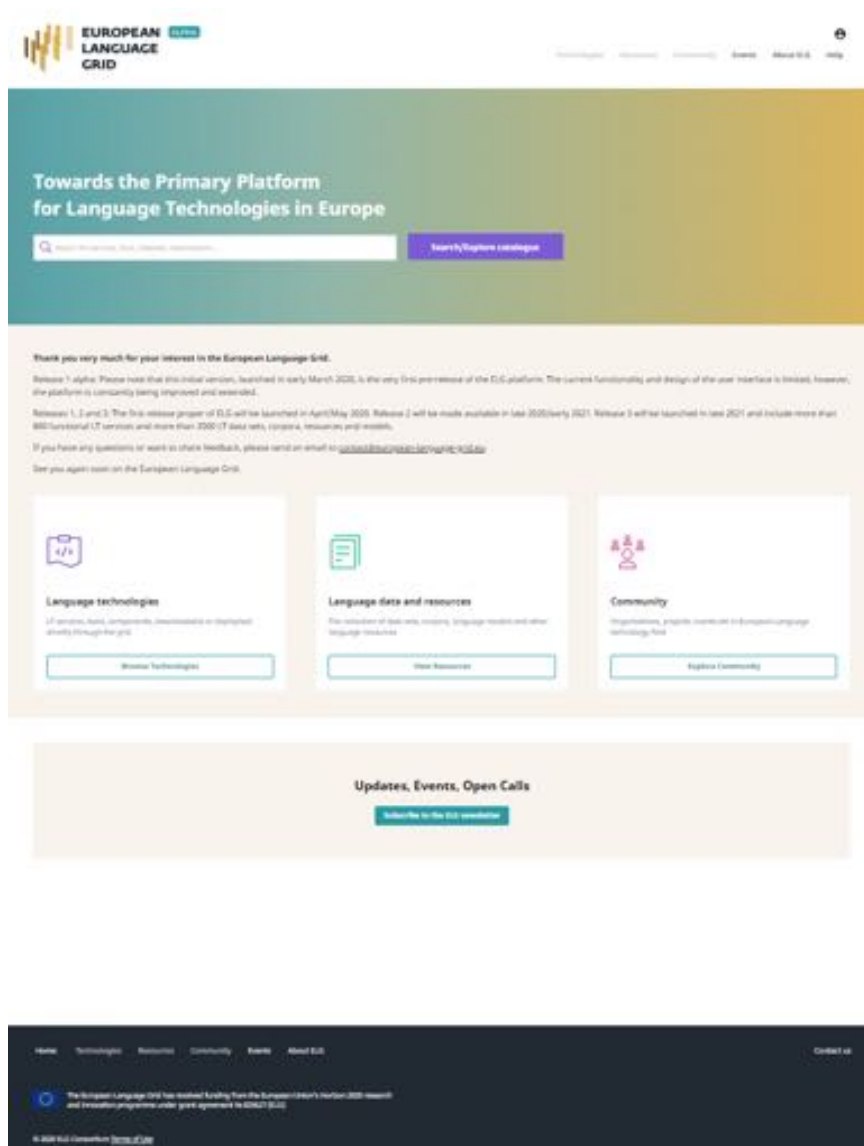


Figure 5. New homepage

When a user reaches the main page, the single sign-on mechanism checks whether the user has been signed on before and, if yes, they are signed in. Some pages are restricted to authorized users only (e.g., upload of resources) and some are not (such as catalogue search results). Depending on whether only authorized users can see the content, and following the initial checking, an unauthorized user may be sent to the login screen.

## 4 Content Management System

### 4.1 Technical Description

Drupal was chosen for static content management, as it is one of the leading and most popular CMS worldwide. Drupal acts as a headless solution, which means that it is mainly used as a back-office application

and end-users do not see anything directly from the CMS. For external access, the Drupal server generates menus and HTML content pages and it facilitates configuration using the REST style API.

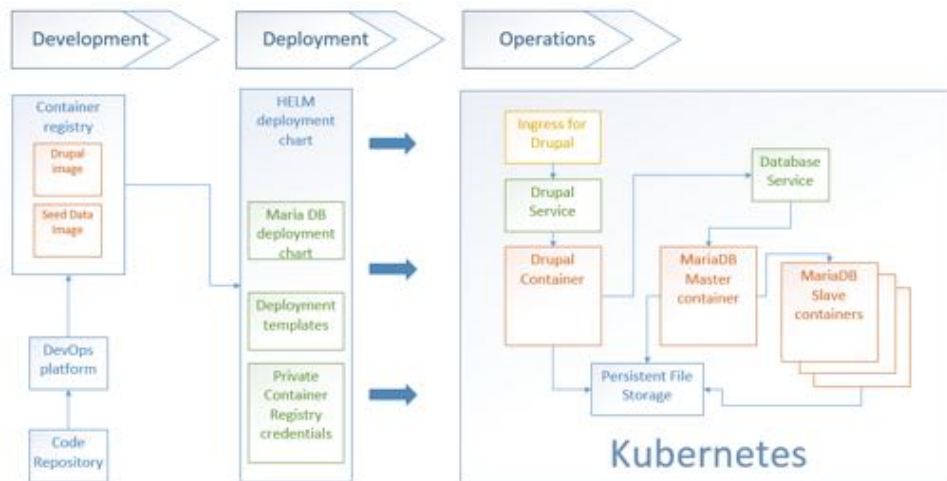


Figure 6. CMS life cycle

Figure 6 describes the CMS development process. It is very similar to the Angular development process: developers upload their code and seed data to the repository, where automatic pipelines push Docker images to the private Azure Container Registry. For the CMS, there are two parts and two Docker images – one for the CMS instance and another – for seed data. The seed data container holds:

1. initial data for the CMS database, e.g., site configuration and HTML content.
2. files for the CMS website, e.g., logos, favicon, images, PDF files and other downloadable files.

The ELG platform deployment pulls these Docker images by providing image pull secrets.

To work for our setup, the Drupal instance needed to be extended with several modules:

- CKEditor: This module allows Drupal to replace text area fields with a visual HTML editor, which brings WYSIWYG editing to the web.
- Menu Item Extras: This module allows adding extra configuration fields for menu items.
- Rest Menu Items: This module allows accessing menu items via REST.
- Keycloak Integration: Integration of our single sign-on solution.
- Webforms and webforms REST module: This module allows the creation of web forms, as well as getting and posting them via REST.

## 4.2 Functionality and Web Interfaces

There are multiple content types served by the CMS. The most basic one is a simple page, for example the landing page (Figure 7). Its content is prepared in the CMS and transferred to the portal in the form of JSON when requested. To be able to create content that looks the same in the CMS and in the ELG portal, a portion of style configuration should be shared between them. A Google Material stylesheet file that is exported from the ELG portal and imported into the CMS ensures common ground between these two components.

Three menus are maintained in the CMS and displayed in the ELG portal. These are the main menu in the header and two secondary menus in the footer – one on the left side that mimics the header menu and one on the right side. There is one link (Contact us) in the right side footer menu at the moment (Figure 8).

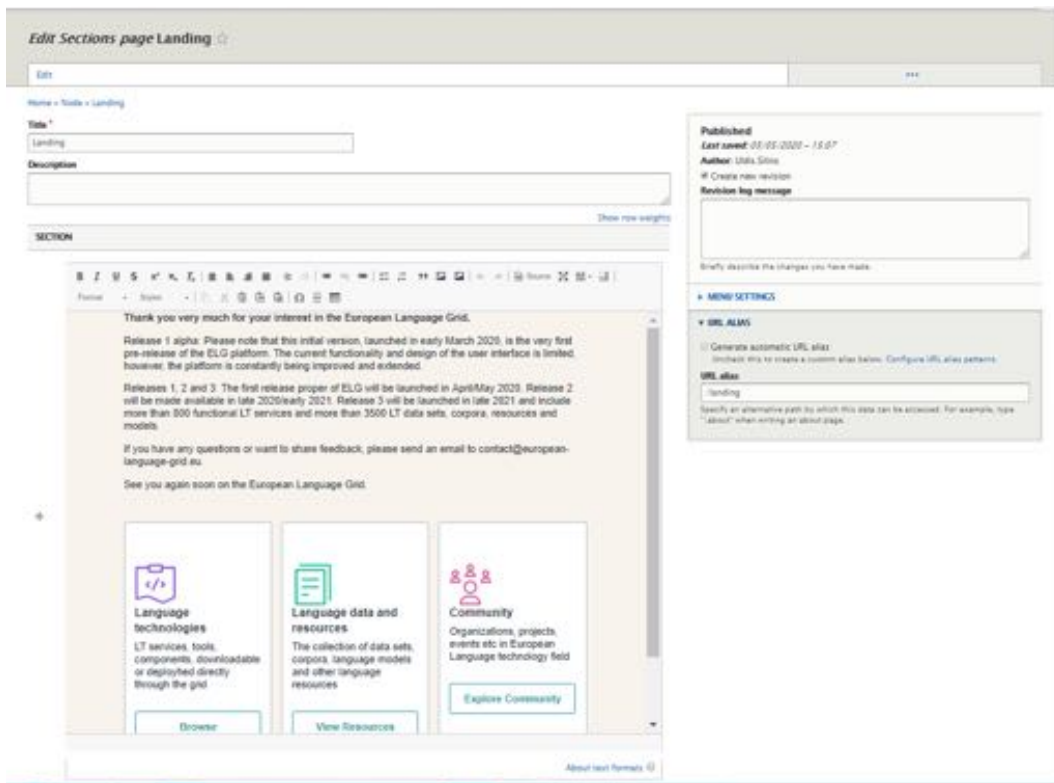


Figure 7. Landing page in the CMS

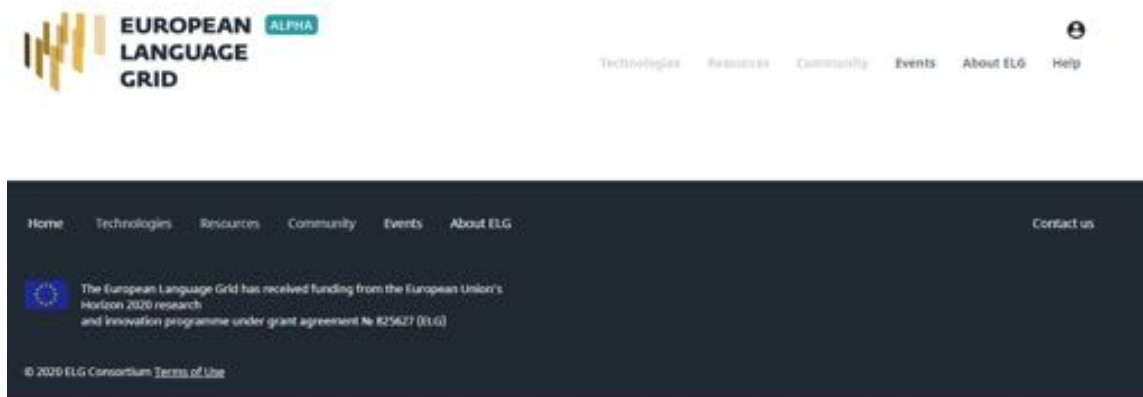


Figure 8. Header and Footer menus



Figure 9. Menu link editing

Each menu item can be configured individually by indicating the target links of the menu item and various other options (Figure 9).

There are additional parameters added to the menu item configuration, such as the ELG menu link type (Figure 10). This is added via the Drupal module “Menu items extras” and it makes it possible to configure the type of link and prepend the URL depending on this type. When the menu information is received by the front-end, there are parameters that mark the type of link (Figure 11). For example, CMS links are prepended with /page/ and React links are prepended with /catalogue/.

Figure 10. Editing a menu type

```
[{"key": "1e4204cc-879d-43c0-b74a-62724657c52f", "title": "Home", "description": null, "uri": "base:home",...}
> 0: {key: "1e4204cc-879d-43c0-b74a-62724657c52f", "title": "Home", "description": null, "uri": "base:home",...}
> 1: {key: "0b9131a4-9f59-41d1-b10f-19434a8f771a", "title": "Technologies", "description": null, "uri": "node/8",...}
> 2: {key: "1755b733-43d6-439a-9e19-3d5114372aaa", "title": "Resources", "description": null, "uri": "node/7",...}
> 3: {key: "53eb13c5-8213-4d32-aa0b-5c27bc72cb93", "title": "Community", "description": null, "uri": "node/15",...}
> 4: {key: "a1807a6e-c429-4c46-a7a4-7946e93a32a4", "title": "Events", "description": null, "uri": "node/13",...}
  key: "a1807a6e-c429-4c46-a7a4-7946e93a32a4"
  title: "Events"
  description: null
  uri: "node/13"
  alias: "conferences-and-workshops"
  external: false
  absolute: "http://dev.european-language-grid.eu/cms/conferences-and-workshops"
  relative: "/cms/conferences-and-workshops"
  existing: true
  weight: "-45"
  expanded: false
  enabled: true
  uuid: "e50ec486-2b42-4fc3-aa6d-0d06e01f0d4f"
  options: {}
  field_elg_menu_link_type: "CMS"
  field_show_as_not_clickable: null
> 5: {key: "2290d108-f6b4-4400-befe-a05a0cf16711", "title": "About ELG", "description": null, "uri": "node/2",...}
```

Figure 11. Menu items received as a JSON object through REST

Access to various CMS actions is granted through a permission mechanism which can be assigned to different roles (see Figure 12). For example, access to menu items is not restricted to logged-in users so that every page visitor can see the menu.

PERMISSION	ANONYMOUS USER	AUTHENTICATED USER	ADMINISTRATOR	CONTENT EDITOR
<b>RESTful Web Services</b>				
Access GET on <i>Menu Tree</i> resource	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
Access GET on <i>Menu items per menu</i> resource	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		

Figure 12. Menu permission mechanism

## 5 Catalogue UI

### 5.1 Technical Description

For the implementation of the catalogue UI, we moved away from the traditional monolith architecture which dictates that back-end and front-end are tightly connected. Instead, we follow the emerging micro-front-end architecture<sup>13</sup> which separates the two. This makes each codebase simpler and easier to maintain. Also, by decoupling the two concepts, the back-end and front-end developers' team can take independent decisions, which makes the development process faster and more versatile. For the development of the catalogue UI we use React<sup>14</sup>, a JavaScript library for user interfaces. The React GitHub repository has more than 145k stars and is widely supported by large enterprises with an MIT license and a vast and vibrant ecosystem.

The catalogue UI is implemented as a Single Page Application (SPA). In this way, the end-user experiences instant page updates instead of full page reloads. This is made possible due to AJAX requests, the Virtual Document Object Model (VDOM) and the Reconciliation heuristic algorithm  $O(n)$  that the library uses to update the DOM of the page. In order to deliver a robust and easy to maintain application, we exploit mature and widely adopted technologies and libraries that are well documented and community supported.

For the state management of the catalogue's data, we use Redux<sup>15</sup> which utilizes the unidirectional data flow pattern. This provides a single place to store and update the data and ensures that each component gets updated whenever a modification in the data is triggered. For the client-side routing, we use react-router-dom<sup>16</sup>. This library provides access to the user's browser history and utilizes the concepts of dynamic routing where routes are rendered while the catalogue UI is rendered. For the authentication and authorization part we use Keycloak's JavaScript library.<sup>17</sup> Following the registration of our application to Keycloak, we are able to access user information (username, e-mail) and acquire authentication tokens. This information feeds into policy-related actions; thus, we can assess if a certain user should be allowed to view restricted pages and if she should be granted access to specific functionalities, like the try out GUIs, for example.

We developed many reusable components for the catalogue UI that are kept as simple and flexible as possible. It promotes code efficiency, maintenance and maximizes reusability of the components.

<sup>13</sup> <https://martinfowler.com/articles/micro-frontends.html>

<sup>14</sup> <https://reactjs.org>

<sup>15</sup> <https://redux.js.org>

<sup>16</sup> <https://reacttraining.com/react-router/>

<sup>17</sup> <https://www.npmjs.com/package/keycloak-js>

One of our primary concerns is to develop and deliver a catalogue that provides a pleasant experience to the end-user independently of the browser or device they are using. In order to have as broad a browser coverage as possible we transform the code into ECMAScript 5 code that is supported by many browser versions. We specifically target a broad range of browsers based on global usage of production builds.

To accomplish these functionalities, we use the official create-react-app front-end build pipeline, which, among others, utilizes webpack for the bundling process and Babel for the transpilation stage. By using this pipeline tool, we also get an optimized production build which is produced each time the GitLab CI/CD is running in order to build the Docker image that we deploy to Kubernetes. This Docker image contains an Apache server which serves our production-ready application.

In order to have some baselines of quality measures for our application we have used Google's Lighthouse<sup>18</sup>, an Open Source auditing tool for performance and accessibility (Figure 13). For upcoming releases, we aim to improve these metrics and focus more on features like SEO and Server-Side Rendering.

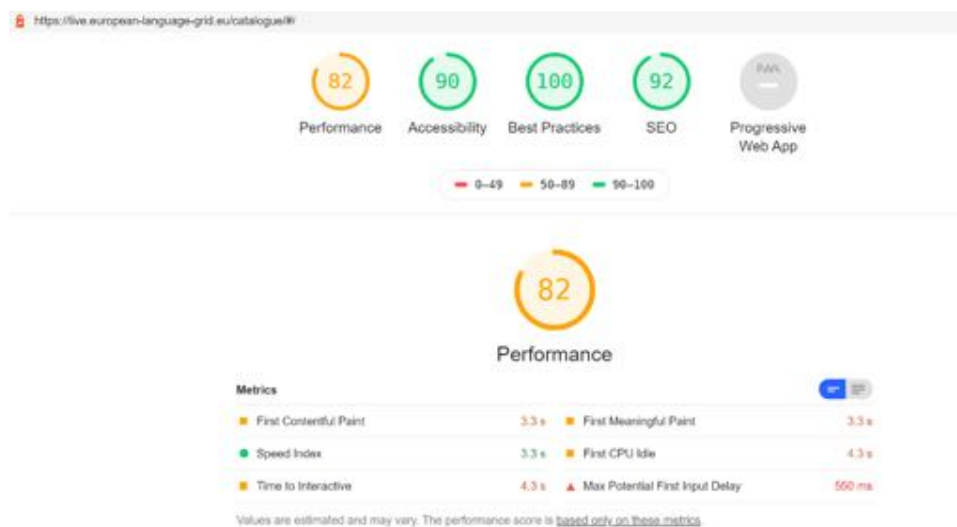


Figure 13. Performance metrics

## 5.2 Functionality and Web Interfaces

The catalogue UI is the main point through which users interact with resources hosted or described in the ELG. These include:

- Functional content, i.e., LT services and applications that are fully integrated into the platform and that can be used through trial UIs supplied by ELG and executed following ELG specifications.
- Non-functional content, i.e., metadata records for
  - data resources (e.g., corpora, datasets, lexica, terminologies, Machine Learning models, computational grammars, etc.) that are available through ELG and/or other catalogues, and
  - tools/services that can only be available in a downloadable form from the ELG platform or that are available through other catalogues or websites
- Metadata records for companies, research organisations, projects, etc. that are active in the LT domain and can promote themselves through ELG, while providing an overview of the LT landscape.

<sup>18</sup> <https://developers.google.com/web/tools/lighthouse/>



ELG targets a wide range of users with different requirements and expectations, broadly divided into the following groups (cf. Deliverables D2.1, D2.3 and D7.1 for more information):

- **content providers** and **developers and integrators** are providers and consumers of LT resources, as described above,
- **information providers** and **information seekers** are providers and consumers of LT-related (meta)-information, as described above,
- **citizens**, i.e., individuals that want inform themselves about LT and that understand the scope of ELG,
- **ELG platform and content administrators**, i.e., the ELG technical team that administers and monitors the day-to-day operation and performance of the platform.

The needs and requirements of each user group have influenced the platform architecture (D2.2), the metadata schema used to describe the entities of the catalogue (D2.3) and, of course, its design.

In view of Release 1 of the platform (due M16), the implementation of the catalogue UI has started but is still at an early stage with a limited set of functionalities, focusing mainly on the consumer's side<sup>19</sup>. The pages we have implemented until now support searching and browsing the catalogue, selecting and viewing descriptions of resources or related entities (organization, project), and testing of functional services. Still lacking is a number of enhancements and features, which are already planned and will be implemented soon (e.g., better rendering of the labels of metadata elements and values, deployment of visual aids for some actions, etc.). We also expect to get feedback from real users of ELG Release 1 alpha, which will be taken into account at the follow-up stages for next releases, if time constraints for this release do not allow immediate reactions.

Keeping in mind the different applications that should be integrated, the catalogue UI follows the overall design concept of the ELG front-end Angular 8 SPA application and corporate identity. Accordingly, it features the same header and footer areas, icons, typeface and colour scheme to achieve a consistent look and feel.

In the following section, we present the main pages of the catalogue UI and their features.

### 5.2.1 Catalogue Browse and Search Page

The catalogue main page displays minimum information, allowing users coming from the ELG main website to get a clear picture of the contents of the ELG platform. The top menu items are inherited from the main ELG website; they redirect the user to the corresponding sections. The structure of the page follows the standard way of presenting resources in other catalogues, enabling users to quickly familiarize themselves with ELG.

The main section of the page lists the catalogue contents in two alternative forms: list of items (Figure 14) and items in the form of cards (Figure 15). The items are sorted by resource type and in alphabetical order; other sorting options will be added at a later stage.

The information displayed for each item is carefully selected to serve as a preview of the full description and to help users decide whether they want to explore the item further. For the time being, and following the main findings of the user survey (D3.1), this information includes: the title of the resource (which is a link to the landing page of the item), a short summary of its description, the licence(s) under which the resource is available, the language(s) of the resource or supported by the tool/service and keywords. Icons on the right are

---

<sup>19</sup> A lot of the functionalities required for the population of the catalogue, such as the upload of metadata records and physical files of resources, management of metadata records and services, are performed through APIs and processes at the back-end side, and are, thus, not presented in this deliverable.

used to provide visual cues or points of reference for users as they move through the interface; for now, only the type of resource or a related entity is used. Other types of information (e.g., popularity, other classification information, etc.) will be added based on user feedback of the item snippet.

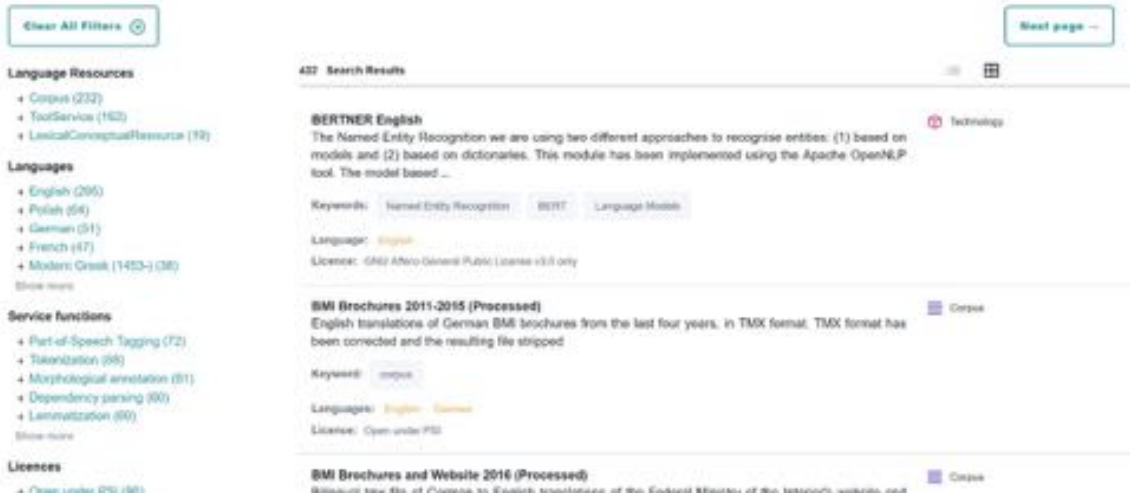


Figure 14. List view of catalogue contents

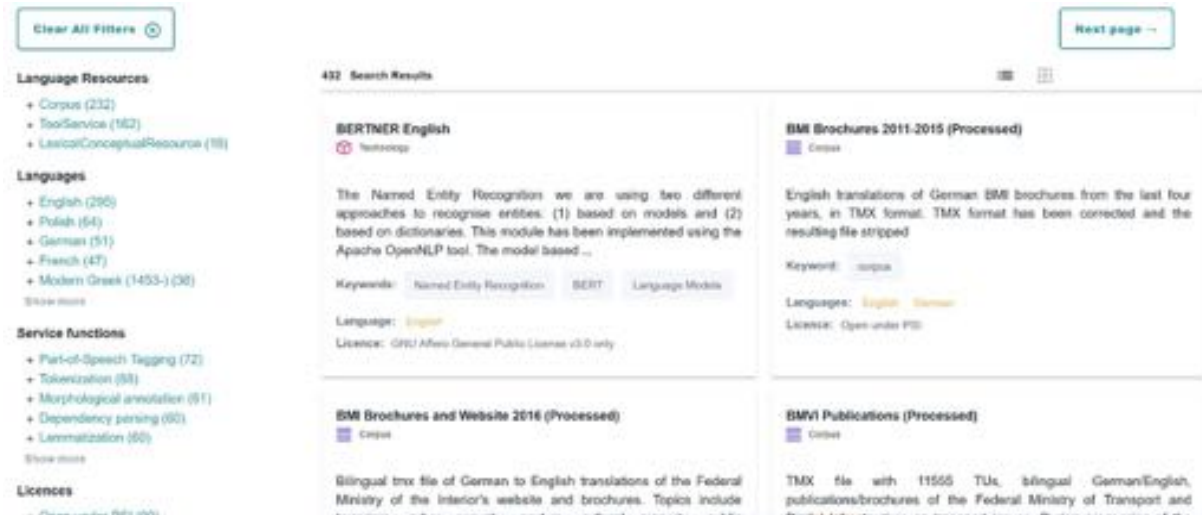


Figure 15. Card view of catalogue contents

Users can browse the catalogue or search for items in two ways: (1) using the bar on top of the page (Figure 16) for free text search, and (2) using the facets (on the left-hand side of the page) to narrow down search results using specific criteria. The search categories both for free text and facets have been carefully selected to reflect user preferences from the survey and are restricted so as not to overwhelm the users with too much information. For Release 1, the facets include language, function for LT services, licence, resource type and related entity type, which are the main features by which users search for resources (D3.1). Again, for future releases, these will be enriched with user requirements.



Figure 16. Header area of the catalogue page

### 5.2.2 Landing Page for Items

Once users have spotted an item they find interesting, they can click on its title and will be directed to a page that describes it in detail. We developed landing pages for each item category (i.e., LT services, data resources and their subtypes, organizations, projects, etc.) included in the catalogue. They allow users to get detailed information on an item, preview its contents or functionality, and, finally, obtain and use it for their purposes.

Even though the landing pages contain information that differ for each category, our main efforts aim to provide a consistent visual look and feel. Landing pages share a common layout that consists of a header, a right-hand sidebar, a main content area and a bottom content region, described in the following subsections. Each page uses the same principles customized for the different categories of content (i.e., Project, Tool/Service, Organization, Corpus etc.). The pages apply the general principles of content organization such as grouping similar items together, numbering items, and using headings and prompt text.

The information displayed on the landing page of each item comes from the respective metadata record. Taking into consideration the specificities and richness of the metadata schema (D2.3), but also user-friendliness, we opted for layering the information along specific sections of the page and highlighting the most important information. Tabs are also used for splitting information into smaller views and enabling users to navigate through the item features: view its general information, try out the service or view the samples of datasets (not yet implemented), view information on how it is distributed and obtain the item (run the service or download the data resource), etc. We also use expansion panels (also known as accordions) to group similar information and keep the landing page at a reasonable length.

The main challenge in designing landing pages for the catalogue resources is that the metadata schema includes a lot of elements that take values from controlled vocabularies, rather than free-text items. This poses a problem since the accumulation of many elements and values together can become tiring to the eye. The proposed design displays the keys (element labels) in small font size and places more emphasis on the values, as we believe that in this way the values are much better identified visually. In addition, the positioning of the elements on the page is carefully thought through to draw users' attention to the most important information.

#### 5.2.2.1 Header

The header area displays the main information for each item. Its purpose is to introduce the user to the current page. The left header region is reserved for the logo of the item (Figure 17); if none is provided by the resource provider or metadata creator, then the default icon associated with its category is displayed (Figure 18).



Figure 17. Header for an organization with the organization logo



Figure 18. Header for an organization with default icon

### 5.2.2.2 Main Content Area

The main region contains the information that is of most interest to users. Since we do not have control over free text items, like, for instance, the description of a resource item, we ensure readability by trimming texts to a maximum of five lines of text. A “Read more” button expands the text to the remaining lines.

The description is followed by the “classification section” (Figure 19) in which we group information that classifies each item according to different parameters, such as keywords, domain, relevant LT area, resource subtype, etc. We use the colour purple to divide the tag-label from the actual tag, and Material’s UI Chips for tag values. We currently investigate if we can use these tags for search queries.

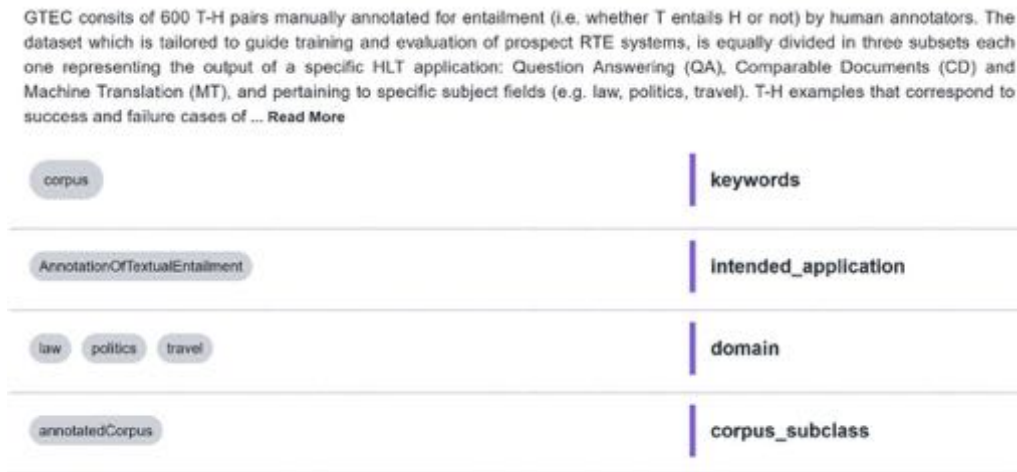


Figure 19. Main section of a corpus

For Projects and Organizations, the main content area is similar. Still, fine tuning for specific information is required. For instance, while the description and classification sections follow the same principles, Organization has a different need for content, i.e., “HasDivision” items (Figure 20). Other categories, on the other hand, include more information and thus additional layouts and styles. One example is the category of Tool/Services.

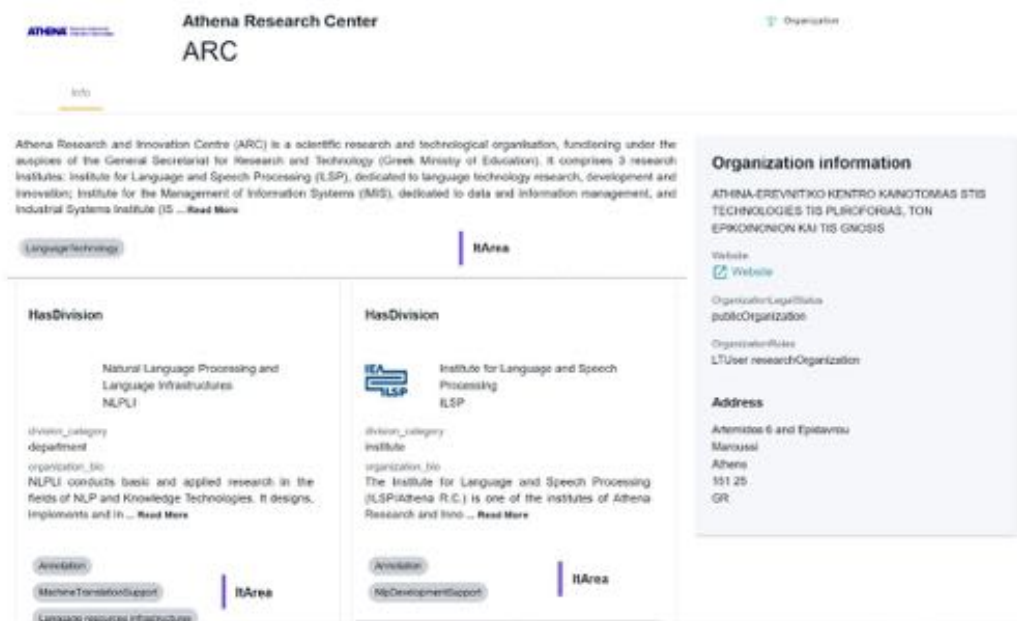


Figure 20. Landing page for an Organization

Figure 21 displays the landing page of a Tool/Service. Below the classification section we use three card blocks to graphically display the actual use of the tool, i.e., the main specifications for the input that the tool expects, the function (operation) it performs and the expected output. In order to keep the cards small, an expansion panel is used that hides some information under a “Read more” link.

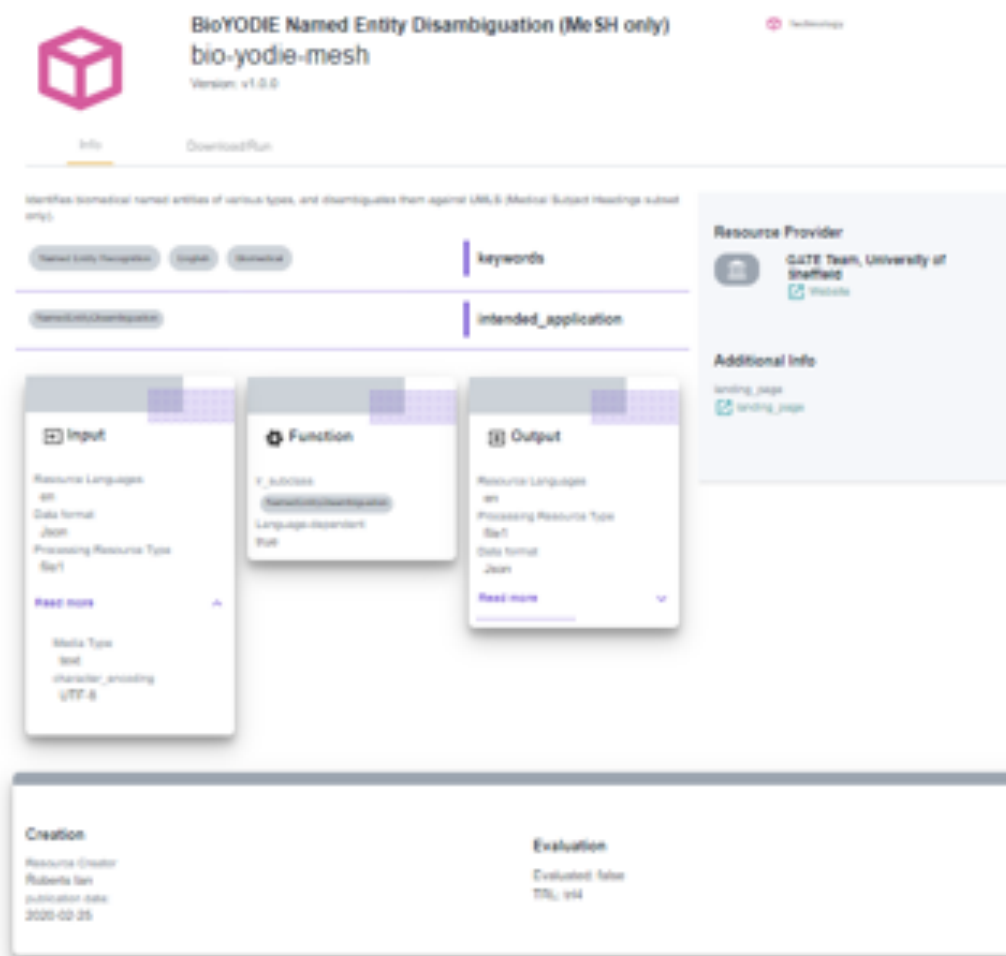


Figure 21. Landing page for a tool/service

In a similar way, the landing page for a Project (Figure 22) also engages users with areas that are visually the same but alters the main content area focusing on a project’s relevant content.

### 5.2.2.3 Sidebar

This area was created in order to show information deemed important for each category while keeping the page short enough so that users do not have to scroll down in order not to miss the information. For instance, for resources, this section includes information on funding projects, contact points where users can ask for more information, etc. The screenshots included in previous sections show the sidebar area, too. A grey background colour distinguishes the sidebar area from the rest of the content.

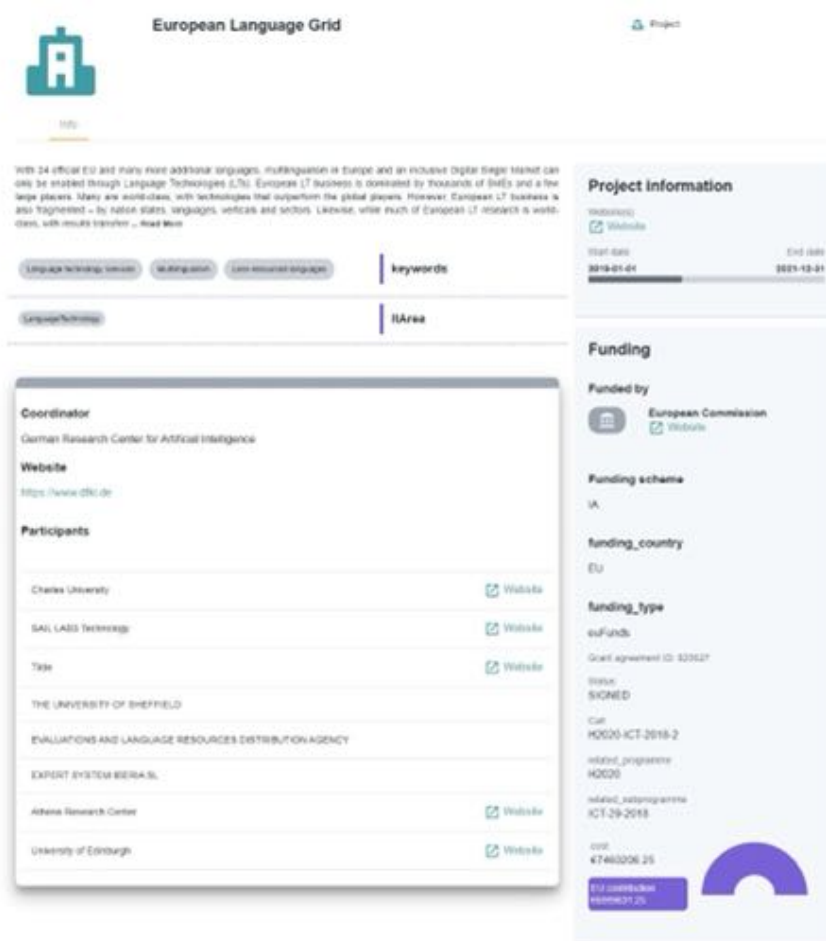


Figure 22. Landing page for a project

#### 5.2.2.4 Bottom Content

The bottom content area (Figure 23) brings together information types that are common to all categories and considered important enough to show on the main tab. Such information includes documents, such as user manuals, publications, etc. where users can learn how to use the resource, have access to information on the resource creator and creation details, evaluation of the resource, etc.



Figure 23. Bottom content area for resources

#### 5.2.2.5 Tabs

Tabs are used to separate information into appropriate groups and draw users' attention to particular details. All categories include a default tab "Info" which contains the main descriptive and technical metadata. Further, different tabs are used depending on the category. Thus, all resources include a tab with information on their

distributable form(s), i.e., the licensing conditions under which a resource may be acquired, technical details (such as format) of the physical files. In later releases, this tab will include the button for running/downloading a tool/service or downloading a data resource. Functional LT services include two more tabs (Figure 24): (1) a “Test/Try out” Tab, which contains the trial UI implemented by ELG so that users can test the service and which is specific to broad categories, namely, Information Extraction, Machine Translation, Speech Synthesis, Automatic Speech Recognition (Figure 25); (2) a “Code samples” tab, which allows end-users with technical knowledge to test a tool in command mode.



Figure 24. Tabs for functional services

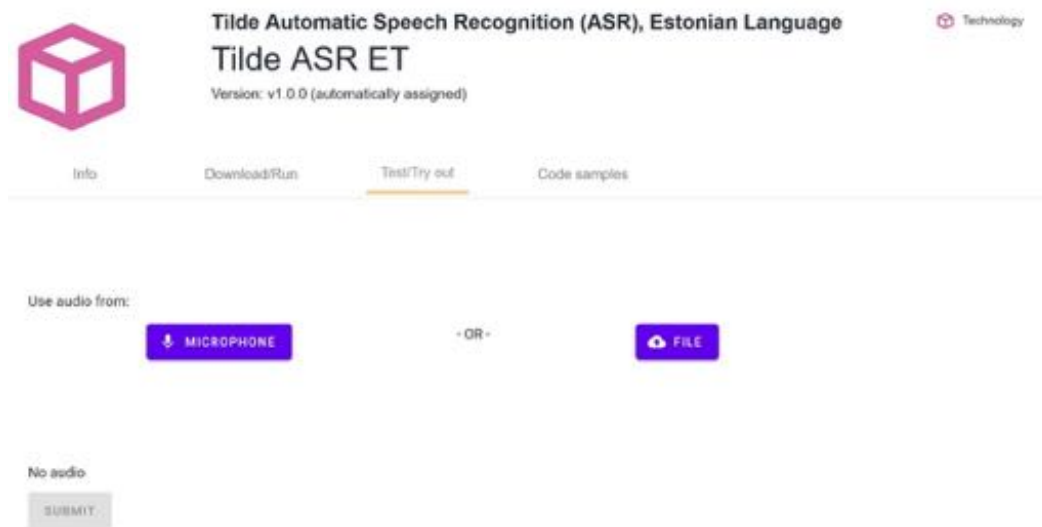


Figure 25. Test/Try out tab for a sample ASR service

## 6 Authentication Solution

### 6.1 Technical Description

For authentication and, partially, authorization functionality, we selected Keycloak<sup>20</sup>, an open source software which is a good fit for the ELG platform. Keycloak provides a wide range of functionality for authentication and authorization. Despite Keycloak’s broad set of features, some catalogue-specific authorization features had to be developed by the ELG team itself. Moreover, the CMS currently uses its own authorization system. However, if necessary, Keycloak authorization capabilities can be used for different ELG parts.

For our needs, a specific realm was created, named ELG. This realm contains a list of clients, i.e., external applications that can do authentication and/or access Keycloak services. Each ELG GUI and back-end component that directly exploits Keycloak have their own client defined: Drupal CMS, Angular application, catalogue React application as well as the Gatekeeper software that controls access to LR services (see D1.3).

<sup>20</sup> <https://www.keycloak.org>

One of the functions of Keycloak most critical for ELG is the centralized user login system. It provides single sign-on capabilities for multiple independent applications using the Open ID protocol that is built upon the open industry-standard protocol OAuth 2.0<sup>21</sup> for authorization. By using the single sign-on capability, after logging into the first application, the end-user will also be automatically logged into other applications. Some front-end applications use the single page application (SPA) approach without a dedicated back-end that could save user sessions (this is essential to also keep the user logged in after page reloading, browser restarting or opening a page in another browser tab). Keycloak saves a user's session state and if the end-user is entering the SPA, the SPA can restore the user's logged-in session using Keycloak. A typical user flow looks as follows:

1. The user accepts the closed area of SPA or clicks on the login link.
2. SPA redirects the user to the Keycloak login page of the ELG realm.
3. Keycloak tries to restore the user session:
  - a. If the session is restored and the user is known, the user is redirected back to the SPA.
  - b. If the user is not known, then the user is asked for their credentials (optionally also external authentication capabilities, like Google, Facebook), and after authentication the end-user is sent back to the SPA.
4. The SPA gets an access token that is sent back together with the end-user redirection.
5. The SPA can extract basic user metadata from that access token: user id, name, surname and e-mail.

One of the currently ongoing development actions is custom theme development. Custom themes allow to design and adjust all GUI forms and e-mails. The current live environment is exploiting Keycloak default themes, but for upcoming releases at least login, password recovery, registration forms and e-mails will be replaced with a custom ELG-based design.

## 6.2 Functionality and Web Interfaces

Keycloak provides default GUIs for both the end-user and for back-office needs such as user management operations and platform configuration. Currently, the following GUIs are available for end-users:

Login (Figure 26): the live environment currently allows registration with locally administered accounts only, however, logins can also be performed with external logins in the development environment.

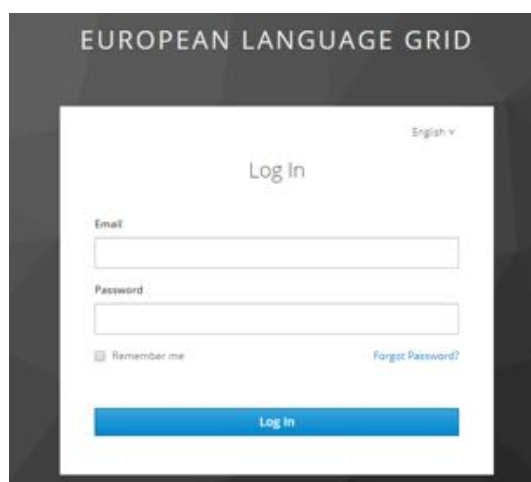


Figure 26. Login form

<sup>21</sup> <https://tools.ietf.org/html/rfc6749> – the OAuth 2.0 Authorization Framework



Registration form (Figure 27): a self-registration form is available right now in the development environment because the public live platform has not been opened for general use yet (interested users have to request a login by sending an email to the ELG administrators).

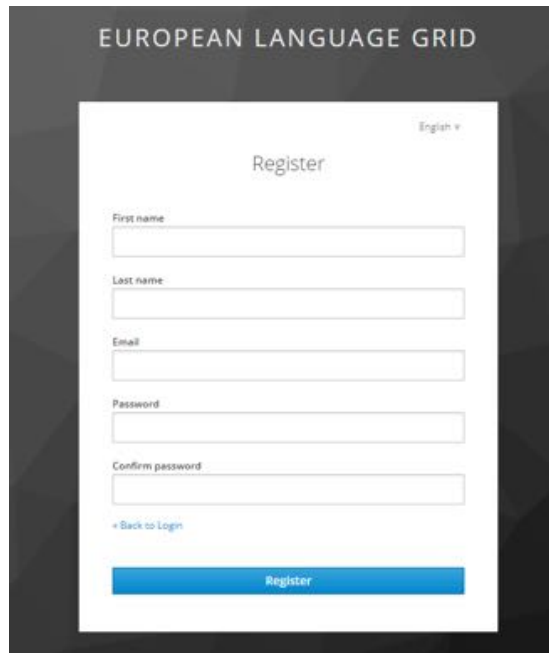


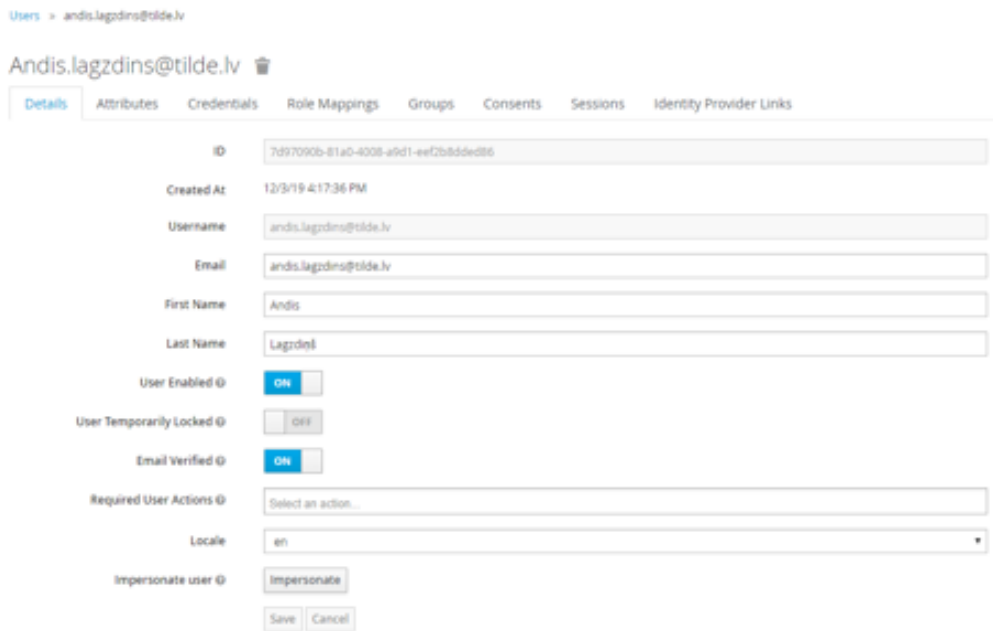
Figure 27. New user registration form

Password recovery form (Figure 28): in case, the user forgot their credentials, they can restore their password by providing the e-mail address that was registered during sign-up. Then, the user will receive an e-mail from Keycloak to reset the password.



Figure 28. Password recovery form

Keycloak provides full support for user management: credentials, management of user details, attributes, lock and unlock user, check access logs, sessions and many more operations (Figure 29).



The screenshot shows the Keycloak user profile form for the user 'Andis.lagzdins@tilde.lv'. The form is organized into several sections with tabs at the top: 'Details', 'Attributes', 'Credentials', 'Role Mappings', 'Groups', 'Consents', 'Sessions', and 'Identity Provider Links'. The 'Details' tab is active, showing the following fields and controls:

- ID:** 7d97090b-81a0-4008-afd1-eef2b8dded86
- Created At:** 12/3/19 4:17:36 PM
- Username:** andis.lagzdins@tilde.lv
- Email:** andis.lagzdins@tilde.lv
- First Name:** Andis
- Last Name:** Lagzdins
- User Enabled:**  ON
- User Temporarily Locked:**  OFF
- Email Verified:**  ON
- Required User Actions:** Select an action...
- Locale:** en
- Impersonate user:**
- Buttons:**

Figure 29. Keycloak user profile form

## 7 Conclusion

In this deliverable we report on the progress of development of the ELG platform GUI. We present an overview of the overall front-end architecture, and provide the technical description and implementation details for the solutions designed for the central portal, CMS, catalogue UI and authentication solution. In February 2021, Deliverable D3.3, “Platform GUI (interim release)”, will provide an update of the current work.