

WP4 INTERLINK Platform

D4.3- First release of INTERLINK platform and community portal



*The INTERLINK project is co-funded by the European Union.
Horizon 2020 - DT-GOVERNANCE-05-2020 - Grant Agreement No 959201*



Project acronym	INTERLINK
Project full title	Innovating governNment and ciTizen co-dEliveRy for the digital sINgle marKet
Call identifier	DT-GOVERNANCE-05-2020
Type of action	RIA
Start date	01/01/2021
End date	31/12/2023
Grant agreement no	959201

WP	4. INTERLINK Platform
Author(s)	Serge Sushkov (TREE), Diego Bernabe (TREE), Cristina Luengo (TREE) Diego López de Ipiña (DEUSTO), Julen Badiola (DEUSTO)
Editor(s)	Serge Sushkov (TREE)
Reviewers	Raman Kazamiakin (FBK), Olli Pelkonen (CNS)
Leading Partner	TREE
Version	V1.0
Deliverable Type	OTHER
Dissemination Level	PU
Date of Delivery	2022-04-30 (M16)
Submission Date	2022-05-11 (M17)



VERSION HISTORY

Version	Issue Date	Status	Changes	Contributor
0.1	12/04/2022	Draft	Initial structure	TREE
0.2	02/05/2022	Draft	1 st version	TREE, DEUSTO
0.3	09/05/2022	Pre-final	Version with reviewers' inputs	FBK, CNS
1.0	11/05/2022	Final	Submitted version	TREE



Glossary

ENTRY	DEFINITION
INTERLINKERs	Common building blocks, provided as software tools or in the form of knowledge offered digitally, that represent interoperable, re-usable, EU-compliant, standardized functionality for the co-production of public services
Public Service	Services that are publicly available and are provided by the government or on behalf of the government's residence in the interest of its citizens. In INTERLINK we focus not only on the software services (i.e., the services delivered digitally) but also the services that rely on digital technologies.
Software Platform	<p>A platform is a group of technologies that are used as a base upon which other applications, processes or technologies are developed.</p> <p>In other words, a platform is the basic hardware (computer) and software (operating system) on which software applications can be run. This environment constitutes the basic foundation upon which any application or software is supported and/or developed.</p> <p>Within the context of the INTERLINK project, we define a Software Platform as a set of data storage, backend services and APIs which serve as a basis for the business logic and frontend applications to develop, integrate and function. It also includes SW deployment and operational infrastructure.</p>
Software Backend	Is part of software services and/or applications running on server side within the client-server paradigm. It mostly dedicates to data storage, business logic, process workflow and utility functions
Software Frontend	Is part of the software services and/or applications running on the client side within the client-server paradigm. It mostly focuses on graphical user interface (GUI), workflow navigations and supporting business logic
SW API	API means Application Programming Interface, a type of software interface, offering a service to other pieces of software.



ACRONYMS

ABBREVIATED	EXTENDED
AAC	Authentication and Authorization
CD	Continuous Deployment
CEF	Connecting Europe Facility
CI	Continuous Integration
DB	Database
DM	Data Model
IT	Integration test
KPI	Key Performance Indicator
OSS	Open-Source Software
PA	Public Administration
SaaS	Software as a Service
SOC	Service Offering Canvas
SSO	Single Sign-On
SW	Software
UT	Unit test
WP	Work Package



Table of contents

1	INTRODUCTION.....	9
1.1	Introduction	9
1.2	Related documents and contents	9
2	INTERLINK PLATFORM AND INFRASTRUCTURE	10
2.1.1.	Software Repository.....	10
2.1.2.	Software Development and Deployment Procedures	11
2.2	<i>Infrastructure Platform Components</i>	15
2.1.1.	Data Storage Layer	15
2.1.2.	User Authentication	15
2.1.3.	Infrastructure Logging	15
2.1.4.	User Activity Logging	18
2.3	Task and Incident Management	22
2.4	The Servers Deployed	23
3	COMMUNITY WEB PORTAL	23
3.2	INTERLINKERS as Enablers of the Co-production Process	24
3.3	Catalogue of INTERLINKERS.....	27
3.4	INTERLINK Collaborative Environment	28
3.3.1.	INTERLINK Collaborative Environment Views	32
4	PREPARATION FOR THE PILOT CASES FOR EVALUATION	35
4.2	Guidelines for instantiation	35
4.1.1.	INTERLINKERS selection per Environment	37
4.3	Specific Instantiations	37
4.1.1.	Latvian Use-Case	37
4.1.2.	Spanish Use-Case	38
4.1.3.	Italian Use-Case	39

List of figures

- Figure 1. Project software repository structure.*
- Figure 2. Infrastructure and User logging architecture.*
- Figure 3. Log messages collected into central LOKI database.*
- Figure 4. Infrastructure monitoring dashboard.*
- Figure 5. User activity monitoring dashboard.*
- Figure 6. Redmine issues list per Work Package and sub-project.*
- Figure 7. INTERLINK API to be integratable in collaborative environment.*
- Figure 8. INTERLINKERS Catalogue data in GitHub repository.*
- Figure 9. INTERLINKER catalogue.*
- Figure 10. Generic co-production model in INTERLINK.*
- Figure 11. Comparison of INTERLINK ENGAGE stage in 2 different co-production projects and INTERLINKER recommendation.*



Figure 12. Comparison of INTERLINK BUILD (standard) vs. equivalent RUN (custom) stages in 2 different co-production projects.

Figure 13. Guide section of the collaborative environment frontend.

Figure 14. Workplan section of the collaborative environment frontend .

Figure 15. Overview view of the Collaborative Environment.

Figure 16. Interlinkers catalogue in the collaborative environment frontend.

Figure 17. Interlinkers catalogue in the collaborative environment frontend.

Figure 18. Environment variables file for demo environment (.env.demo).

Figure 19. Reduced metadata.json file for Business Model Canvas knowledge INTERLINKER.

Figure 20. Reduced environment file for Latvian use case

Figure 21. Result of the customization variables applied to the Spanish use case

Figure 22. Reduced environment file for Spanish use case

Figure 23. Result of the customization variables applied to the Spanish use case

Figure 24. Reduced environment file for Italian use case

Figure 25. Result of the customization variables applied to the Italian use case

List of tables

Table 1. Example of log format for tracking of C.R.U.D. operations on Asset entity.

Table 2. Example of log format for tracking inside of INTERLINKER modules.

Table 3. Co-production INTERLINKER API.



Executive summary

This deliverable represents the first release of INTERLINK platform, including the community web portal, catalogue of digital enablers for the collaborative process (INTERLINKER modules), as well as the software development environment and deployment procedures, including guidelines for instantiation of the three use-case pilot demo servers and specific instantiation instructions.

First, the software repository structure, software development and software deployment procedures are described in **Section 2.1**.

Then the major infrastructural platform components, namely, the User Authentication service and Infrastructure and User Activity Logging are described in **Section 2.2**.

Sections 2.1 and 2.2 are placed under the common Chapter 2 reflecting the Task T4.3 within the Work Package (WP) 4 of the INTERLINK Project.

Description of the implemented INTERLINK Community Web Portal is reported in separate **Chapter 3** as it corresponds to Task T4.4 of the WP4 of the Project.

Chapter 4 of this document describes the software installation and preparations for the demonstrations and testing of the three pilot use cases of the Project. These activities correspond to the tasks under Work Package 5 and their results will be reported in a separate deliverable document.

1 Introduction

1.1 Introduction

INTERLINK is designed as a collaborative software system, which consists of numerous software components.

This document describes the first stage of this integration, in which all single components are integrated into a collaborative system. The architectural basis has been explained in the previous deliverable D4.2, where we described the principles and concepts of the overall system architecture in detail. Hence, this document will show how they apply to provide the expected result in the INTERLINK platform.

1.2 Related documents and contents

During the project, many conceptual and architectural deliverables have been created, which build the basis for the INTERLINK platform. The following enumeration lists the most relevant of them. Each of them describes aspects, which apply to a single component or idea in much greater detail. We will refer to those documents where necessary, notably:

- D3.1 (FBK, R, M10) - Identification and specifications of INTERLINKERS. Specifications of common building blocks for INTERLINK inclusive public services and their specification.
- D4.2 (TREE, OTHER, M12) - Reference architecture model and specification. The reference architecture model and specifications as defined in T4.1.
- D5.1 (DEUSTO, R, M12) - Use-case plans and guidelines v1. Result of T5.2, this document contains the specification of the use-case plan, including purpose and background, objectives and evaluation criteria, strategy, prerequisites, assumptions, risks, personnel and responsibilities, organisation, site description, methodology, schedule and test result collection. It also describes the associated trial evaluation plan and KPIs. Two releases are planned, one for each phase.
- D5.2 (VARAM, R, M12) - Community building and preliminary use-cases activities. Result of T5.3, this document contains the plan for building a community for the users and stakeholders in all the use-case sites, including details about the communication channels and contents.
- D2.1 (RU, R, M16) - Preliminary governance model. This report will include a literature review and a preliminary governance model identifying relevant variables and conditions. The model will also take into account the comparative analysis of successful and unsuccessful cases of co-production.
- D2.3 (RU, R, M16) - Governance performance indicators. This document will be a list of operationalized, non-technical performance indicators, to be used in T5.2 to develop KPI for the evaluation of the platform.



- D2.4 (CNS, R, M16) – Co-business model specification and analysis. This report will briefly describe the alternative co-business models considered for INTERLINK, present analysis results identifying the strengths and weaknesses of each candidate model and specify the best co-business model to be supported by the INTERLINK platform.
- D3.2 (FBK, OTHER, M16) – Initial repository of INTERLINKERs and partnership tools. This deliverable will provide an initial repository of common core INTERLINK enablers (INTERLINKERs) to foster Government as Platform model, and of public-private partnership governing tools such as partnership models, templates, and guidelines. The initial repository will cover a subset of the enablers targeted at the first use-case validation.

2 INTERLINK Platform and Infrastructure

First, the software repository for the INTERLINK Project had been established on GitHub.com server. This satisfies the requirement for the project software to be free of commercial licensing and open source.

The project's GitHub repository has been structured according to the structure and types of the software in the project. Next, the software development and software deployment policies had been elaborated. Finally, the software hosting environment had been defined and first deployments started.

2.1.1. Software Repository

The following structure has been established for the project software repositories in GitHub and is reflected in the naming convention for the software repositories:

- Interlink-project: the main configuration and documentation for the entire project software release;
- Frontend: the frontend part of the collaboration web portal;
- Backend-X: various backend components, both for the web portal and platform type services;
- INTERLINKER-X: software services corresponding to the INTERLINKER building blocks.

The Figure 1 below presents the GitHub repository of the project, which could be also seen online at <https://github.com/orgs/interlink-project/repositories>.

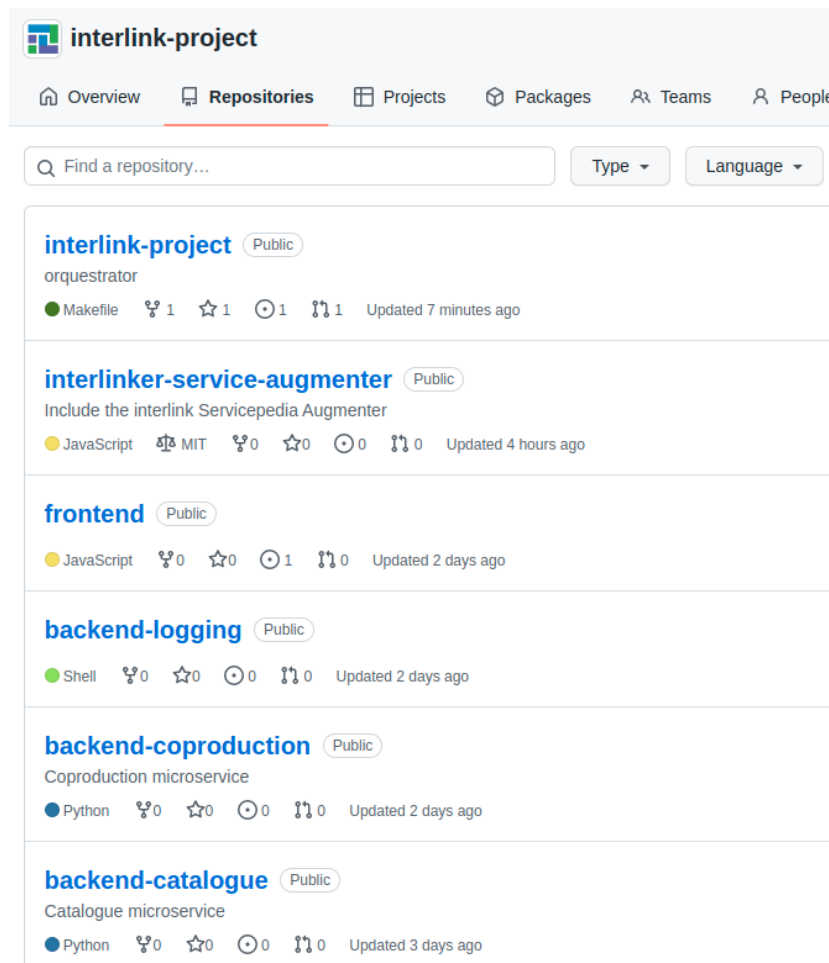


Figure 1. Project software repository structure.

2.1.2. Software Development and Deployment Procedures

The following software development and software release management have been established for the project software.

Continuous Integration (CI)

Unit Tests (UT)

Every software package should have Unit Tests (UT) provided by software package authors. Project DevOps should configure an automatic workflow to execute those UT on every push to the master branch.

Software Releases

Once a software component achieves an important implementation milestone by developers, its code should be TAGged. For software version which are intended for



deployment, a software release should be built per each software repository. Either assigning a TAG or having built a software release should trigger automatic building of a new docker image(s) for this software repository. Docker images should contain TAG or software versions somewhere in the name of the docker image.

All INTERLINK software packages should be grouped into a single INTERLINK software release, with a way to configure which versions (TAGs) of particular software packages should be included in a given version (TAG) of the INTERLINK software release. For example, INTERLINK software release version 2.1-zgz consists of the following software versions per software package:

- Frontend v1.8
- Backend v3.2
- Zgz module v2.6
- etc.

Integration Tests (IT)

Once the entire INTERLINK software release is tagged, this should trigger execution of Integration Tests (IT) to check compatibility between cross-dependent software packages within the software release.

Building of Docker Software images

Once both UT and IT are completed in "green", i.e., on successful (OK, without errors) termination of UT + IT pipelines, a new pipeline building docker images should launch.

Having no UT/IT available, the pipelines to build docker images should be triggered by assignment of TAGs to software repository or by having built a software release.

Continuous Deployment (CD) & Environments

There might be the following SIX environments:

- Local
- DEV
- Staging (= DEMO)
- pilot MEF
- pilot ZGZ
- pilot VARAM

Local Environment

It is a local computer of a software package developer. He/she can deploy any versions (branches, tags) of software there in order to develop his/her software package. This environment has no docker images autocompiled at GitHub repository, no CI/CD pipelines, etc. and it goes completely under the responsibility of the particular developer himself/herself.



Development Environment (DEV)

This environment is used for manual tests of all the software functionalities, integrations, etc for developers of all the Project software packages. For this, docker images should be re-built per every new push to master branches by CI pipelines once all UT/IT passed. Once docker image(s) built successfully, CD pipelines are started to deploy those new docker images into DEV server.

Staging Environment (DEMO)

This environment is used for non-developer members of the project for demonstrations and GUI testing and feedback purposes. For this, docker images are built by CI pipeline out of latest TAGs assigned per every software package, having all packages passed UT/IT successfully. The DEMO server is intended for testing and training sessions with business users, and their work should not be suddenly interrupted by a new docker image deployment. That's why, once the docker images with new staging software versions are built, DevOps admin should announce to testing users and somehow coordinate with them day and time of deployment of the new software to the staging (DEMO) server.

Pilot Servers (ZGZ, MEF, VARAM)

Deployment of INTERLINK software to pilot servers should contain the common INTERLINK base software (backend, frontend and other platform type components like DBs, auth, etc) and pilot-specific components (ZGZ software components for ZGZ pilot, etc.). The deployment process should be similar to deploying on staging (demo) server.

Software Updates

Minor updates (patches)

Minor software updates are needed when there is either a bug fix or small functionality improvement which (almost) does not affect (or affects very little) other software components, API interfaces, Data Model, etc. Usually, the reason for such software update appears during software testing and corresponding issue is created in Redmine for corresponding software component. Fixing the software code should start a.s.a.p. New software release with minor version incremented (for ex. 3.2.1) should be created. Deployment should be done on the first possible occasion (e.g., overnight or a weekend).

Such software change may happen within the same software release, for example, during the first pilot demo sessions.

Major Software Updates (new Software Release)

Such code update corresponds to implementing a new significant functionality. New software release should be built and deployed with a new major version (e.g., 3.2 or 3.2.0). Usually, the new software release should keep back-compatibility, the same



software design, interfaces, system architecture and data models should be preserved, but could be extended.

Such software change should happen per different project milestones, for example, there will be initial and final pilot demonstrations with different software releases.

Software Refactoring

This is a very rare case when system architecture, API interfaces and/or data model are changed significantly or recreated anew. For API this would imply new documentation and testing, for DB this would imply data migration from old DBs into the new ones, with corresponding DB data export and import scripts, etc.

Such software changes usually should not happen within the lifetime of the same project. It may happen when a new project is started as a continuation of another one.

Docker-compose profiling

Docker-compose profiling is a useful mechanism to structure and group lower and upper level software services within docker-compose YAML file. It is described in detail here: <https://docs.docker.com/compose/profiles/>

The idea is to try to have separate docker-compose files per each software service and to include (or exclude) them into the deployment of a particular environment.

Data persistency

For the first pilot demos the configuration data is read out from JSON files stored in GitHub software repository. The user activity data (new co-production processes, task state changes and asset instances) are kept as long as the DB docker container is not re-installed. On DB container re-deployment these data may be lost.

Current situation

If DB is part of the software release (e.g. web portal) and requires re-deployment on re-installation of the web portal, then to allow bugfix software updates during the pilot demo sessions but at the same time to keep the user activity data, corresponding data export and import should be realized by use of additional data export/import scripts.

Separation of Platform Software Services

Another approach would be to have low level platform services like DB separated from web portal and other components, so re-deployment of web portal (either backend and/or frontend) should not re-deploy the DB container. In this way, the same DB continues to run keeping the accumulated data.

Data backups

Backing up data is a good practice independently on the DB container configuration and re-deployment policies. Having data backed up periodically will save from the global crashes at the level of the hosting server, as well as from incidental data loss



during e.g. maintenance or other activities. Backed up data should be kept on a physically different server, ideally, on a different hosting data centre location.

2.2 Infrastructure Platform Components

The platform components are those backend software services which carry out very basic and common services for the other project software, for example, databases, user authentication, logging and so on. All those services and their features are described in details in D4.2. As mentioned there, the orchestration of the software components is based on the Docker containers per each software service component, hosted in the cloud virtual servers.

This document reports on what has been deployed in the v1 of the three pilot demonstration cases.

2.1.1. Data Storage Layer

The data storage layer is implemented by deploying MongoDB, PostgreSQL and Redis databases as separate Docker containers, but combined in the common docker-compose. yaml configuration file. These database containers may have a common network disk mounted for purposes of data archiving (if needed).

Configuration data currently is persisted in the GitHub repository in the form of JSON files and is read out into databases at the web portal start-up phase.

Currently standalone web services use their own instances of databases in separate Docker containers. If needed, they may use separate databases within common merged Docker containers (one per MongoDB cluster, another for PostgreSQL, another for Redis, etc.)

2.1.2. User Authentication

The Authentication and Authorization Control (AAC) software service is adapted for use in the INTERLINK Project: user authentication within the co-production web portal is integrated with AAC. It has been implemented according to the specification presented in detail in D4.2 and has been deployed as a standalone software service.

2.1.3. Infrastructure Logging

The main goal of the Infrastructure Logging System is to monitor health and operations of all the software components and to alert INTERLINK software administrators on possible problems with the running software Services (e.g., disk full, severe errors in software logs, etc.).



Infrastructure logging stack software collects, stores, processes and visualises data about general functioning of all the other software components at infrastructure level, i.e., Docker containers and software services (like HTTP servers, DB engines, etc) within the containers.

Both infrastructure and business level (user activity) logging and monitoring have similar data flow consisting of:

- Collection of logs data: either from application level or from docker containers;
- Centralised storage of logs data: log messages from all the applications and docker containers are stored in the central logging storage DB;
- Processing of stored logs data: either a basic common processing (indexing, profiling, grouping, etc.) or a customised KPI-specific processing of logs data is performed;
- Data report visualisations: a configurable web dashboard which presents various types of processed data.

Because the infrastructure and user activity logging have many functionalities in common, single common architecture has been elaborated for them, as depicted in Figure X below.

The green part on the left of Figure 2 is the Infrastructure Logging Software Stack. The blue in the centre is the Collaborative Web Portal. And the custom User Activity Logging software components are in red colour.

As an initial version of User Activity Logging, the user activity data captured at the Web Portal Frontend may be streamed into the Infrastructure Logging Software Stack (for example, HTTPd server can write such messages to a separate log file, listened by Promotail log published, etc.) and visualised in its back-office web dashboard without any custom-made processing of users' activity data. Such implementation is included for the first pilot demonstrations.

For the final version of User Activity Logging, own custom data processing, data storage/retrieval and own visualisation dashboard should be implemented and integrated into the Web Portal Frontend.

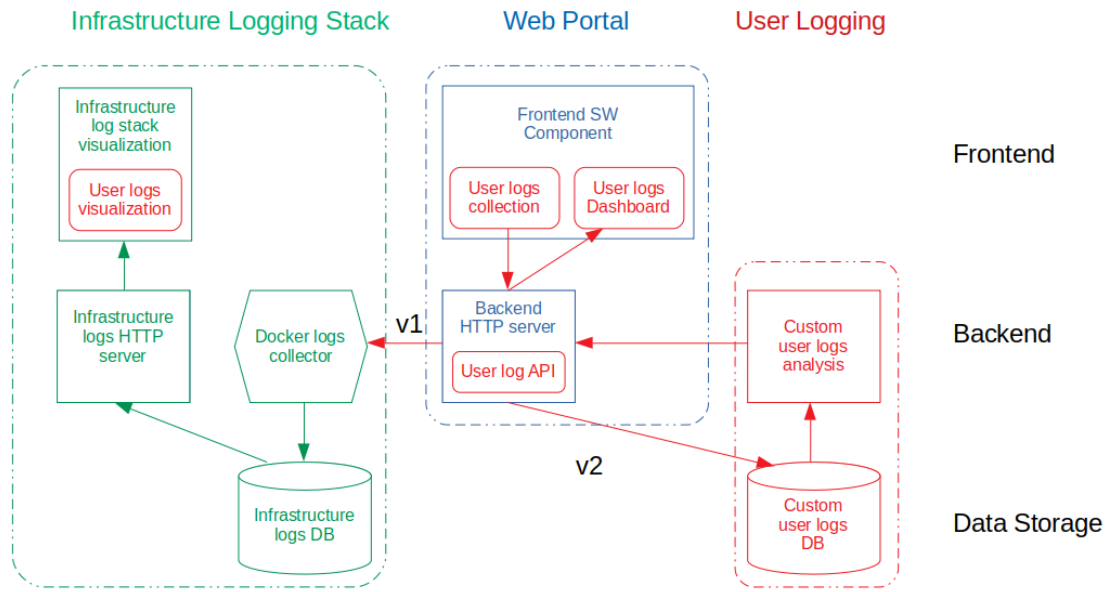


Figure 2. Infrastructure and User logging architecture.

For the infrastructure logging and monitoring, the Grafana+Prometheus stack (<http://Grafana.com>) has been configured and deployed.

To collect log messages from Docker containers, Promotail daemon is configured to run within each container, reading log messages from corresponding log files and sending them to the central logging DB server which is implemented by LOKI software (<https://grafana.com/oss/loki>).

Figure X below illustrates log messages collected into the common logging database LOKI server.

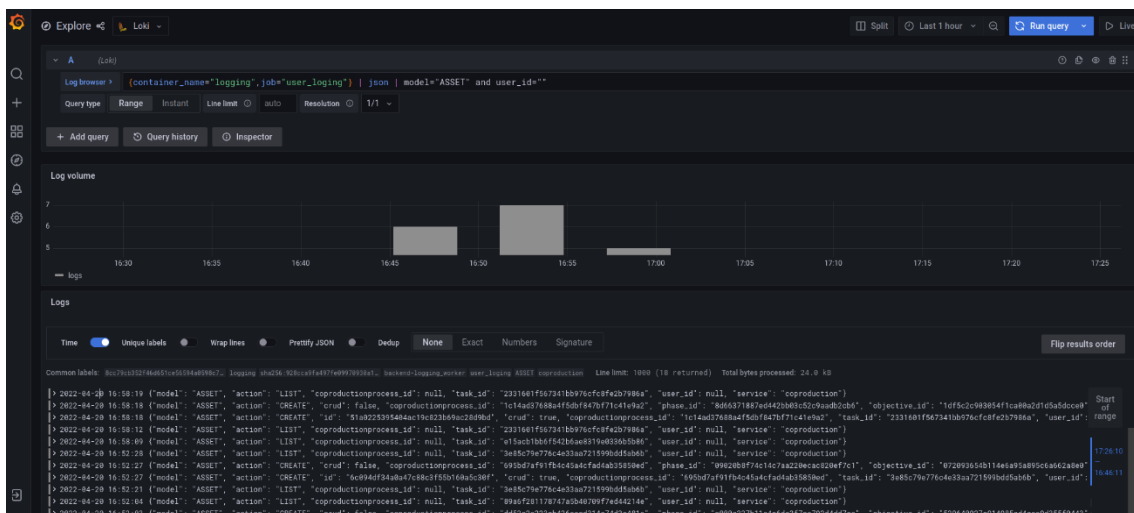


Figure 3. Log messages collected into central LOKI database.



Figure X below shows the web dashboard visualizing the infrastructure log monitoring as implemented for the first pilot demonstrations. This dashboard allows to keep under control the set of the software modules deployed and being ran.



Figure 4. Infrastructure monitoring dashboard.

2.1.4. User Activity Logging

Behavioural logs arise from the activities recorded when users interact with a computer system. In general, user behaviours of interest can include:

- low-level actions such as the keystrokes used when interacting with a productivity application;
- the content viewed in web browsers or e-readers;
- the search queries and result clicks captured by web search engines;
- browsing patterns and purchases on e-commerce sites;
- content generated and shared via social media;
- the history of edits to wikis or other web documents, etc.

Some of the above data is primary (web clicks, search query strings, etc.), but others are KPIs calculated by a data processing engine over the primary and post-processed data (for example, browsing patterns, purchase associations, etc.).

In the case of INTERLINK public service co-production web portal, the primary data which we can log should track user participation in the co-production processes, in particular , user navigation and actions within the web portal areas:



- Public service co-production process flow, with Data Model entities
 - Co-production process
 - Phase of co-production process
 - Phase Objective
 - Task within the Phase Objective
 - INTERLINKER which operates within current Task on an Asset
 - an Asset
- User and Team management, with DM entities
 - User,
 - Team within co-production process
 - User Role within a Team
- Catalogue of INTERLINKERs, with DM entities
 - INTERLINKER description entry in the Catalogue
- Catalogue of Public Services
 - Public Service.

During the PS co-production process, users will navigate within the collaborative environment web portal, as well as use some platform-type or even external software Service INTERLINKERs. Thus, logging of user activity will be implemented inside the code of the two different software:

- the web portal frontend;
- and the platform-type software INTERLINKERs (those integrated within the web portal).

Technically, user activity logs are generated in the following 2 ways:

- at the backend code of the main web portal functionality API, capturing corresponding CRUD (Create, Read, Update, Delete) functions on the core Data Model entities;
- By making HTTP call to the new logging API which has been developed explicitly for the logging purposes: it allows to make HTTP calls from the web portal frontend pages as well as from inside the INTERLINKER software module web pages.

Table 1 below shows example data fields which may be logged at the level of operations on an Asset:

Table 1. Example of log format for tracking of C.R.U.D. operations on Asset entity.



Log message field	Field comments
User_ID	ID of user within the SSO auth system
DateTime	Data and time of the logging event
LogTag	UserAct_ASSET
CoProdProcess_ID	ID of Co-Production Process
CoProdPhase_ID	ID of Co-Production Phase
CoProdPhaseObjective_ID	ID of Phase Objective
CoProdObjTask_ID	ID of Objective Task
Cat_INTERLINKER_ID	ID of a record on given INTERLINKER in the Catalogue
Run_INTERLINKER_ID	UNIX-like Process_ID (e.g., autoincremental BigInt) of user session for that INTERLINKER (between start and finish of visualising it in the web portal frontend)
CoProdAsset_ID	ID of Asset being created, read, updated or deleted
CoProdAsset_Type	Type of an Asset (text, spreadsheet, calendar, etc.)
Asset_Action	Type of action: C.R.U.D. and additional types if needed within the context (e.g., CLONE as creating an Asset as a copy of another Asset, to distinguish from creating a new Asset from scratch)
Specific_Payload	Placeholder to use for context-specific JSON data (e.g., ID of another Asset being copied)

For the user activity logging inside those integrated INTERLINKER software modules (i.e. for detailed tracking of what user does inside that INTERLINKER), additional generic user logging API will be created. This API will consist of only POST method because sending logs is a one-direction process (no need to read, update or delete logs themselves here).

Within various INTERLINKER software modules (those which are of platform-type, i.e., integrated into the web portal), there may be generic fields which are common to all INTERLINKERS (like User_ID, INTERLINKER_ID, Action, etc) and fields specific to functionality of particular INTERLINKER (e.g. Discussion_thread_id, etc).

The table below presents an example of data to log from inside INTERLINKERS.

Table 2. Example of log format for tracking inside of INTERLINKER modules.

User_ID	ID of the user generating logged action
DateTime	Date and time of the log message
LogTag	UserAct_INTERLINKER_Internal



Cat_INTERLINKER_ID	ID from catalogue of INTERLINKERs (static for given INTERLINKER)
Run_INTERLINKER_ID	UNIX-like Process_ID (e.g. autoincremental BigInt) of user session for that INTERLINKER (between start and finish of visualising it in the web portal frontend)
Internal_Action	Type of action from a fixed list of possible actions (e.g. {Navigating (reading) web page, launching / stopping INTERLINKER, creating / using an asset file, etc. })
Asset_ID	ID of an Asset if being created or used
Asset_type	Type of an Asset if being created or used
JSON_PAYLOAD	JSON payload with INTERLINKER-specific details data

Once the logging data is collected and stored in the central logging DB, its basic processing is carried out automatically, for example:

- indexing of log messages by tags, DateTime, ID of users, Co-Production Process, type of INTERLINKER module, type of asset, etc.;
- profiling of data per major field like date, user_ID, CoProdProcess_ID, type of INTERLINKER, type of asset, etc.
- aggregation of data per field, for example, total, min, max and average of users, actions, Co-Production Process, tasks, INTERLINKER modules, assets per day, user_ID, Co-Production Process, task, INTERLINKER, asset, etc.

Having all this basic processing done, the monitoring administrator configures the web dashboard to visualise the basic logging data, aggregated values and time series. Figure 5 below presents visualisation of some of the user activity logging data.

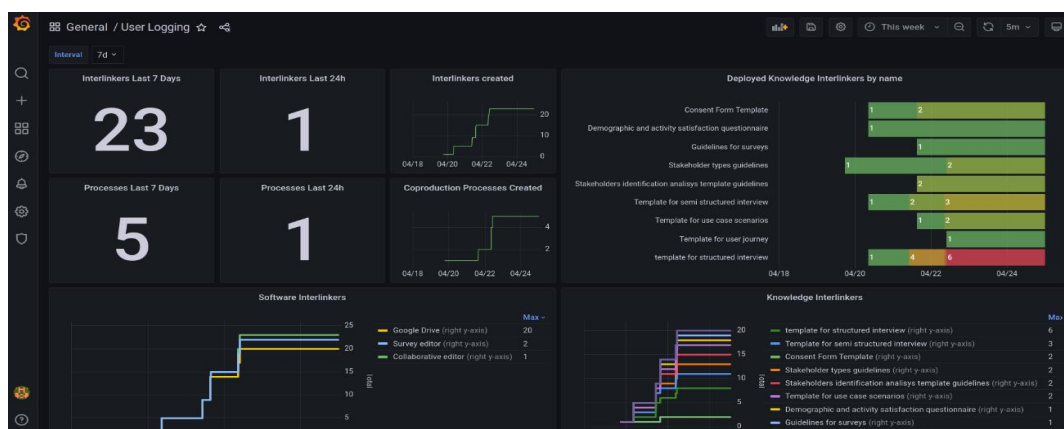


Figure 5. User activity monitoring dashboard.



2.3 Task and Incident Management

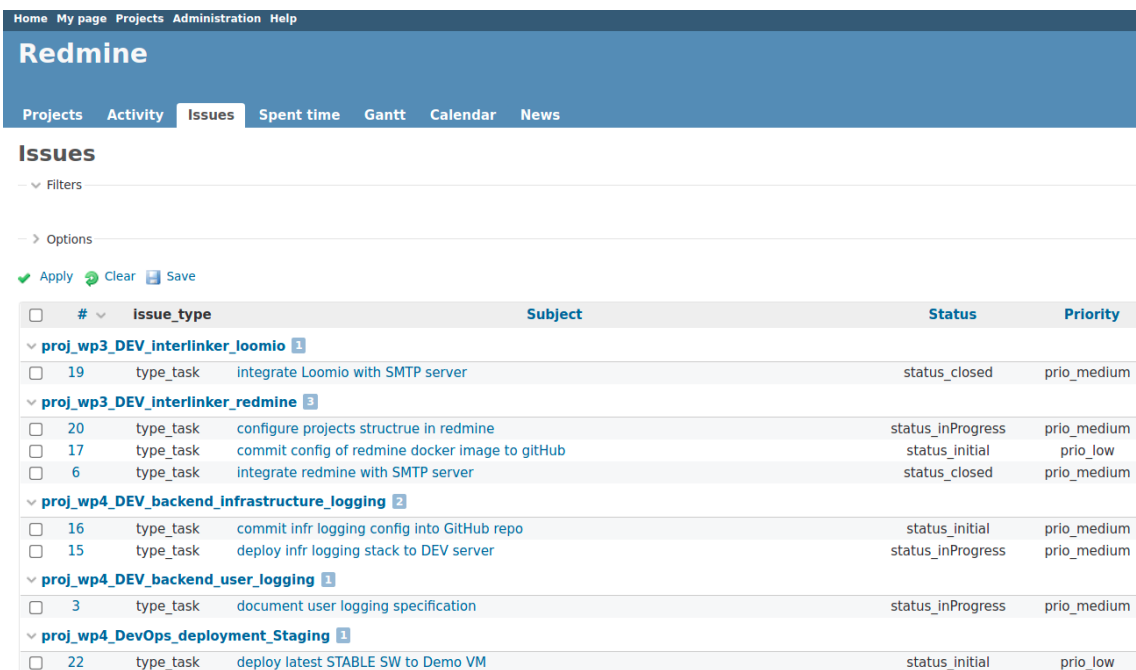
For the task and incident tracking within the project, a single and common Redmine server has been deployed and configured. The server is configured reflecting the structure of the INTERLINK Project by Work Packages, in particular:

- WP3 for development of INTERLINKER modules;
- WP4 for the web portal and the platform;
- WP5 for the demonstration sessions, etc.

The following three types of activities can be tracked with the Redmine server:

- Incidents: when some misbehaviour is discovered in the operation of the web portal and any of the involved software;
- tasks: particular work actions planned or proposed;
- feature: a new functionality or an improvement being proposed, which may be converted into a planned task later.

Figure 6 below illustrates one of the views in the Redmine portal, listing the existing issues per Work Package and sub-projects.



The screenshot shows the Redmine 'Issues' page. At the top, there is a navigation bar with 'Home', 'My page', 'Projects', 'Administration', and 'Help'. Below this is a secondary navigation bar with 'Projects', 'Activity', 'Issues' (selected), 'Spent time', 'Gantt', 'Calendar', and 'News'. The main heading is 'Issues', followed by 'Filters' and 'Options' sections. Below these are buttons for 'Apply', 'Clear', and 'Save'. The main content is a table of issues with columns for '#', 'Issue type', 'Subject', 'Status', and 'Priority'. The issues are grouped by project/work package, with expandable sections for each group.

#	Issue type	Subject	Status	Priority
proj_wp3_DEV_interlinker_loomio				
19	type_task	integrate Loomio with SMTP server	status_closed	prio_medium
proj_wp3_DEV_interlinker_redmine				
20	type_task	configure projects structure in redmine	status_inProgress	prio_medium
17	type_task	commit config of redmine docker image to gitHub	status_initial	prio_low
6	type_task	integrate redmine with SMTP server	status_closed	prio_medium
proj_wp4_DEV_backend_infrastructure_logging				
16	type_task	commit infr logging config into GitHub repo	status_initial	prio_medium
15	type_task	deploy infr logging stack to DEV server	status_inProgress	prio_medium
proj_wp4_DEV_backend_user_logging				
3	type_task	document user logging specification	status_inProgress	prio_medium
proj_wp4_DevOps_deployment_Staging				
22	type_task	deploy latest STABLE SW to Demo VM	status_initial	prio_low

Figure 6. Redmine issues list per Work Package and sub-project.



2.4 The Servers Deployed

The mid-term project demonstrations for the three pilot use cases are described in detail in Chapter 4 below. The following five servers have been hosted and deployed with the INTERLINK software according to the types of the operational environment:

- DEV server: for internal usability and integration tests;
- STAGING (DEMO): for a stable version the web portal for business-level tests and training processes (<http://demo.interlink-project.eu>).
- Pilot / ZGZ: Zaragoza pilot demonstration server (<http://zgz.interlink-project.eu>).
- Pilot / MEF: Italian pilot demonstration server (<http://mef.interlink-project.eu>).
- Pilot / VARAM: Latvian pilot demonstration server (<http://varam.interlink-project.eu>).

3 Community Web Portal

The INTERLINK collaborative environment has been designed to support the co-production methodology of INTERLINK and facilitate its adoption and application in the co-production of novel public services. The portal front-end allows to foster INTERLINK collaborative environment, combining standard exploration and interaction with the information and tools delivered in WP2 and WP3 and according to the requirements elicited in T4.1. The INTERLINK Collaborative Environment offers for a given co-production process alternative views, namely Overview, Workplan and Guide views. Through such views, it provides user access partnership and collaboration tools, guidelines and templates and most importantly to the core building blocks through the INTERLINKERS developed in T3.2.

It offers the following core functionalities:

- a) co-producer team and project management
- b) guide for co-production process, a “how to” guide to take these partnerships towards a successful deployment of new co-delivered public services.
- c) recommendation of INTERLINKERS most suitable to the problem profiles represented by the chosen co-production task
- d) selection, instantiation and registry of use (displaying result of using the enabler, e.g., instantiation of a Business Plan) of a given INTERLINKER. The instantiation of an INTERLINKER, no matter if it is a software or knowledge one, usually gives place to a new resource which contributes to the completion of a co-production process task.
- e) INTERLINKER catalogue where imported INTERLINKERS and co-produced ones are published.

3.2 INTERLINKERs as Enablers of the Co-production Process

INTERLINKERs, as already specified in D3.2, are common building blocks, provided as software tools or in the form of knowledge offered digitally, that offer interoperable, reusable, EU-compliant, standardised functionality for public service co-production management. These enablers are designed to support the co-production of effective, participatory, and sustainable public services. They can be applied to the following purposes:

- *To guide co-production*: Co-production enablers that guide and support teams in the collaborative execution of the co-production initiatives.
- *To build capacity*: Partnership tools and knowledge resources, which tackle the legal, social, and business aspects to make co-delivered public services viable and feasible in time.
- *To aid service development*: Technical enablers for co-delivered services, aligned with other existing EU-wide initiatives to foster interoperable and sustainable public services.

Some examples of *software INTERLINKERs for co-production* are: a) Tools for ideas crowdsourcing and collaborative decision making; b) Tools for surveys; c) Tools for team management; d) Document sharing & File management tool. On the other hand, some exemplary *knowledge INTERLINKERs for co-production* are: a) Guidelines and canvas to perform stakeholders analysis; b) Templates for stakeholders' engagement plan; c) Templates for surveys for problem refinement; d) Guidelines and materials for workshops for service design or e) Templates for Business Plans. Some exemplary *knowledge INTERLINKERs to build capacity* are: a) Guidelines on GDPR for Data Protection; b) Information sheets and consent forms; c) Guidelines on the acquisition and reuse of software for public administrations. Some exemplary *software INTERLINKERs supporting service building* are: a) Registration and authentication component; b) Collaborative Editor for public service descriptions; c) Loyalty, incentives, and rewards component.

In order to support the continuous growth of a catalogue of INTERLINKERs to empower the co-production process, a *Specification Model for INTERLINKERs* has been defined.

The INTERLINKER specification model aims at classifying INTERLINKERs across different dimensions to guide and support the co-production process activities, comply with standards, and foster reuse. Each INTERLINKER must supply a set of metadata in the form of several categories. Regarding *usage*: a) problems it addresses; or b) Service offering type in EU CEF SOC model. Regarding *licensing*: Software and Data licences. Regarding *context*: a) Administrative: local, national, EU; b) Regulatory: standards, regulations it complies to; c) Organisational: PA, Business, Individuals as beneficiaries and d) Domain: application domains, cross-cutting concerns. Regarding *software*: a)



Provisioning: SaaS, OSS; b) Interoperability; c) Security: protocols and d) Integration within the platform.

Following a design pattern similar as the one defined in Research Object Crates (RO-CRATE), INTERLINK has defined an extensible declarative model, based on JSON Schemas, to easily define new either knowledge or software INTERLINKERS. The way to add new INTERLINKERS is to create a new directory per INTERLINKER that contains:

- A "*metadata.json*" file in the root of the directory.
- Optionally, a "*snapshots*" directory to store the images corresponding to the INTERLINKER.

Knowledge INTERLINKERS usually contain several representations of the template, e.g. document (docx), spreadsheet (xlsx), presentation (pptx) and so on, from which it will be instantiated so that users may view what capability they offer before instantiating them. Besides, they often include an *instructions.md* file which explains its usage.

Software INTERLINKERS usually contain a *logo.png* file to be able to depict them in the collaborative environment. Besides and very importantly, apart from common metadata to all enablers (e.g., problem profiles targeted, difficulty, licence, name, description, etc.) they also include aspects to enable its integration with the collaborative environment, whenever they are of co-production type, e.g., through the "*capabilities*" dictionary which includes elements such "*instantiate*", "*clone*", "*view*", "*edit*", "*delete*", "*download*" or "*open_in_modal*" Boolean fields among others. Fig. 7 illustrates the corresponding API methods to be provided by every software INTERLINKER to be neatly integrated with the collaborative environment. On the other hand, Fig. 8 shows the GitHub repository where all INTERLINKERS that populate the Collaborative Environment, and more concretely its Catalogue, are published, following the mentioned Specification Model.

¹ «Research Object Crate (RO-Crate)», Research Object Crate (RO-Crate). <https://www.researchobject.org/ro-crate/>.

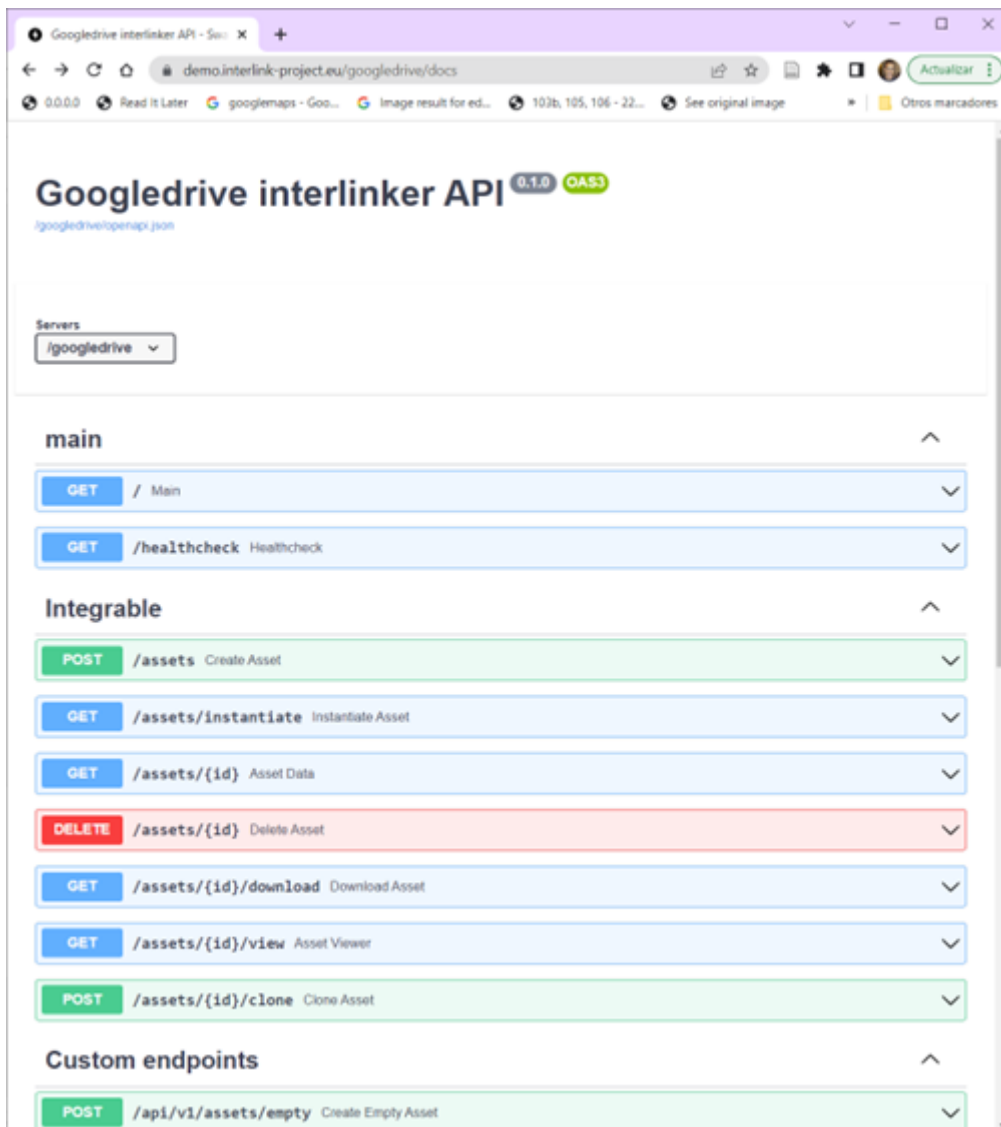


Figure 7. INTERLINK API to be integratable in collaborative environment.

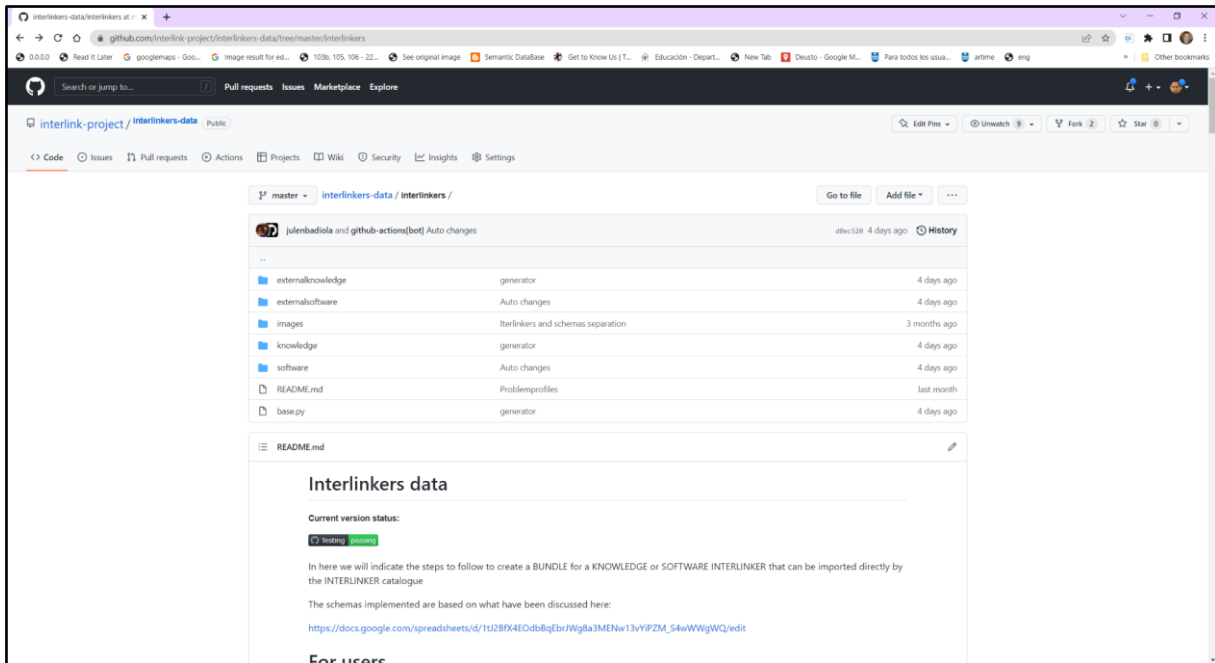


Figure 8. INTERLINKERs Catalogue data in GitHub repository.

3.3 Catalogue of INTERLINKERS

The INTERLINKER catalogue provides a one-stop-shop for know-how enabling co-production. It has been populated with knowledge and software INTERLINKERs leveraging resources generated in previous EU projects, social innovation initiatives, and service design best practices like: WeLive, Silearning.eu, servicedesigntools.org, DesignersItalia, IDEO or Engage2020. Some resources have been adapted to the specific needs of co-production; others are being created from scratch based on project research results. Fig. 9 shows the INTERLINK catalogue where items can be filtered according to strings associated to their metadata, to their nature (software or knowledge), who created them and their ranking.

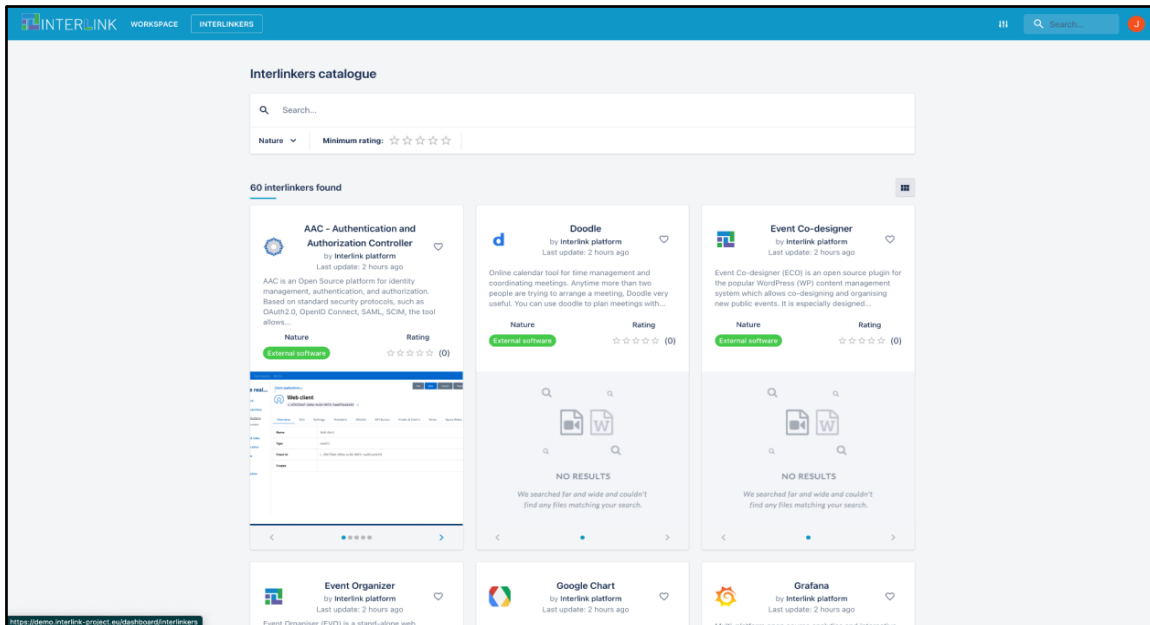


Figure 9. INTERLINKER catalogue.

3.4 INTERLINK Collaborative Environment

The INTERLINK collaborative environment has been designed to support the co-production methodology of INTERLINK (see Fig. 10) and facilitate its adoption and application in the co-production of novel public services. As previously mentioned, it offers the following core functionalities: a) co-producer team and project management; b) guide for co-production process; c) recommendation of INTERLINKERs most suitable to the problem profiles represented by the chosen co-production task; d) selection, instantiation, and registry of use (displaying result of using the enabler, e.g. instantiation of a Business Plan) and e) INTERLINKER catalogue already showcased in Fig. 9.

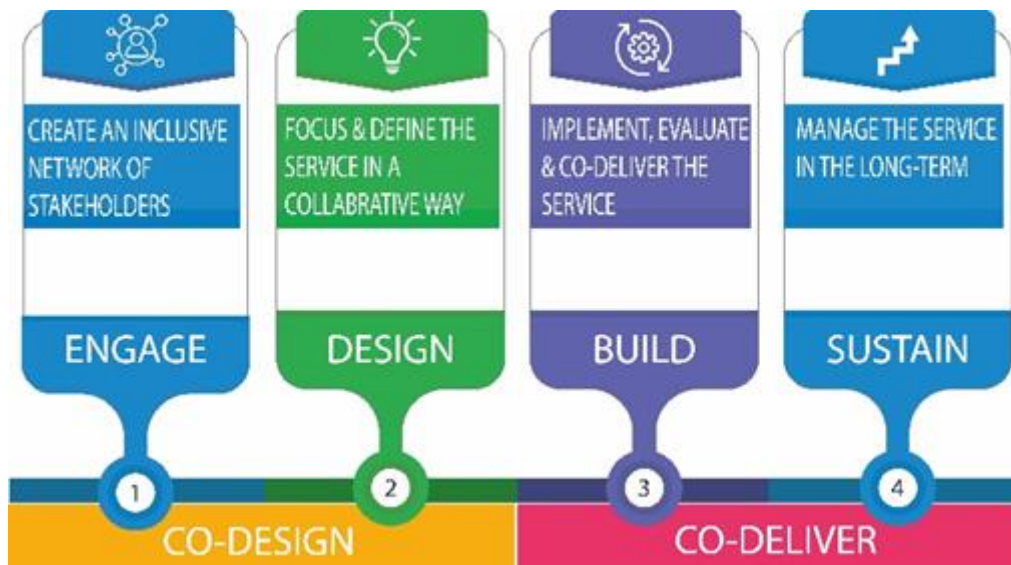


Figure 10. Generic co-production model in INTERLINK.

Notice that apart from methods required to integrate a co-production INTERLINKER with the collaborative environment, see Table 3, custom endpoints are defined by each INTERLINKER, e.g. for GoogleDrive the endpoint shown as /api/v1/assets/empty (see Fig. 8).

Table 3. Co-production INTERLINKER API.

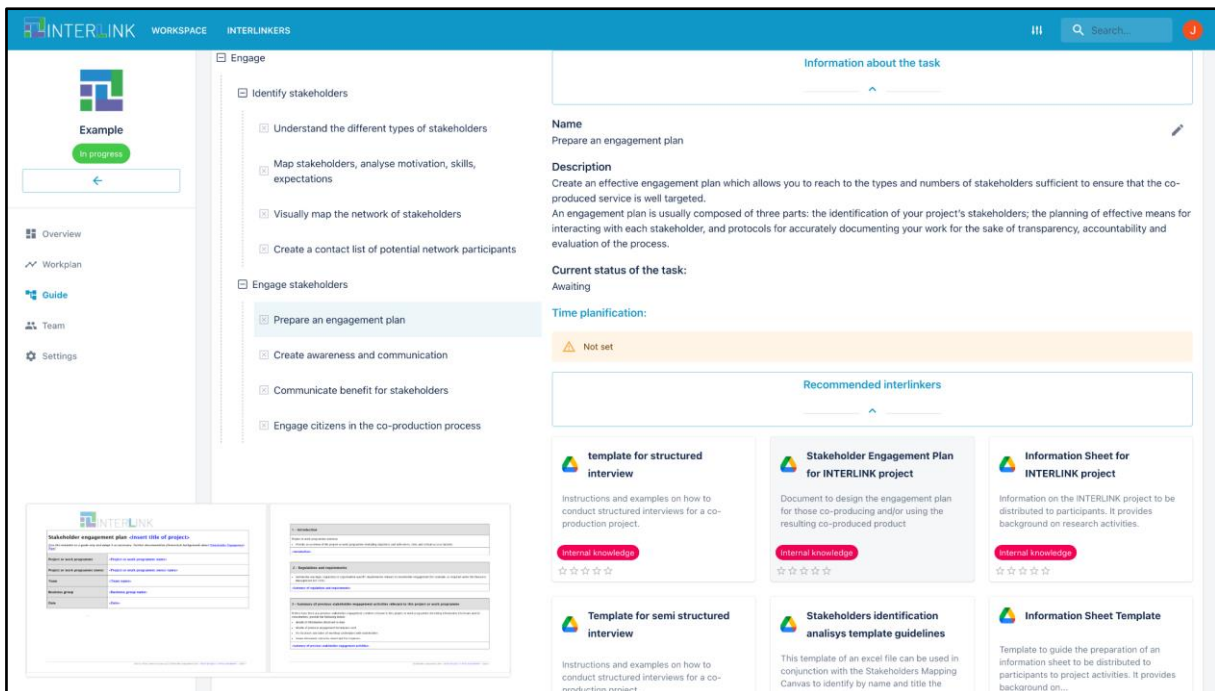
URI	Method	Description
/	GET	redirects to swagger / redoc DOCS
/assets	POST	[OPTIONAL] Posts data for asset creation and return JSON of asset
/assets/instantiate	GET	GUI for asset creation
/assets/{ASSET_ID}	GET	JSON data of asset
/assets/{ASSET_ID}	DELETE	Deletes asset and returns No content
/assets/{ASSET_ID}/download	GET	Download a representation of asset
/assets/{ASSET_ID}/view	GET	GUI for the interaction with the asset
/assets/{ASSET_ID}/clone	POST	[OPTIONAL] Clones the asset and returns JSON data

An assortment of co-production INTERLINKERs has been created to provide useful functionality to the collaborative environment, e.g.: a) interlinker-google drive to deal



with office like documents; b) interlinker-survey to design and host answers for surveys; c) interlinker-ceditor to collaboratively edit documents or d) description augments to annotate web pages.

As already mentioned, JSON Schemas have been defined to declaratively define Software and Knowledge INTERLINKERS. Likewise, co-production models can be defined which are tuned to the specifics of a co-production process, e.g. a Hackathon organisation and celebration. Indeed, although the collaborative environment is pre-loaded by default with the generic INTERLINK co-production tree, applicable in any co-production process, see Fig. 10, purpose specific co-production trees can be defined as shown in Fig. 11 and Fig. 12. Notice that Fig. 13 shows the INTERLINKERS recommendation capability of the collaborative environment, where the same task in two different co-production trees has been selected, recommending the same INTERLINKERS plus additional specific ones for the second co-production tree. Fig. 14. shows how the generic build sub-phase is replaced in the custom hackathon's co-production tree by a run sub-phase, with very different composing objectives and tasks.





The screenshot shows the 'ENGAGE' stage of a project. The left sidebar includes a navigation menu with 'Overview', 'Workplan', 'Guide', 'Team', and 'Settings'. The main content area is titled 'Engage' and contains a list of tasks: 'Identify stakeholders', 'Engage stakeholders', and 'Engage citizens in the co-production process'. The 'Engage stakeholders' task is selected, showing a detailed view with fields for 'Name' (Prepare an engagement plan), 'Description', 'Current status of the task' (Awaiting), and 'Time planification' (Not set). Below this, there are three recommended interlinkers: 'template for structured interview', 'Stakeholder Engagement Plan for INTERLINK project', and 'Information Sheet for INTERLINK project'. Each recommendation includes a brief description, a star rating, and a red 'Internal knowledge' tag.

Figure 11. Comparison of INTERLINK ENGAGE stage in 2 different co-production projects and INTERLINKER recommendation.

The screenshot shows the 'BUILD' stage of a project. The left sidebar is similar to the previous screenshot. The main content area is titled 'Build' and contains a list of tasks: 'Technical implementation', 'Service implementation', and 'Service codelivery'. The 'Service codelivery' task is selected, showing a detailed view with fields for 'Name' (Build), 'Description', 'Current status of the phase' (Awaiting), and 'Time planification' (Not set). The description text is more extensive than in the previous screenshot, detailing the process of finding reusable building blocks and monitoring development.

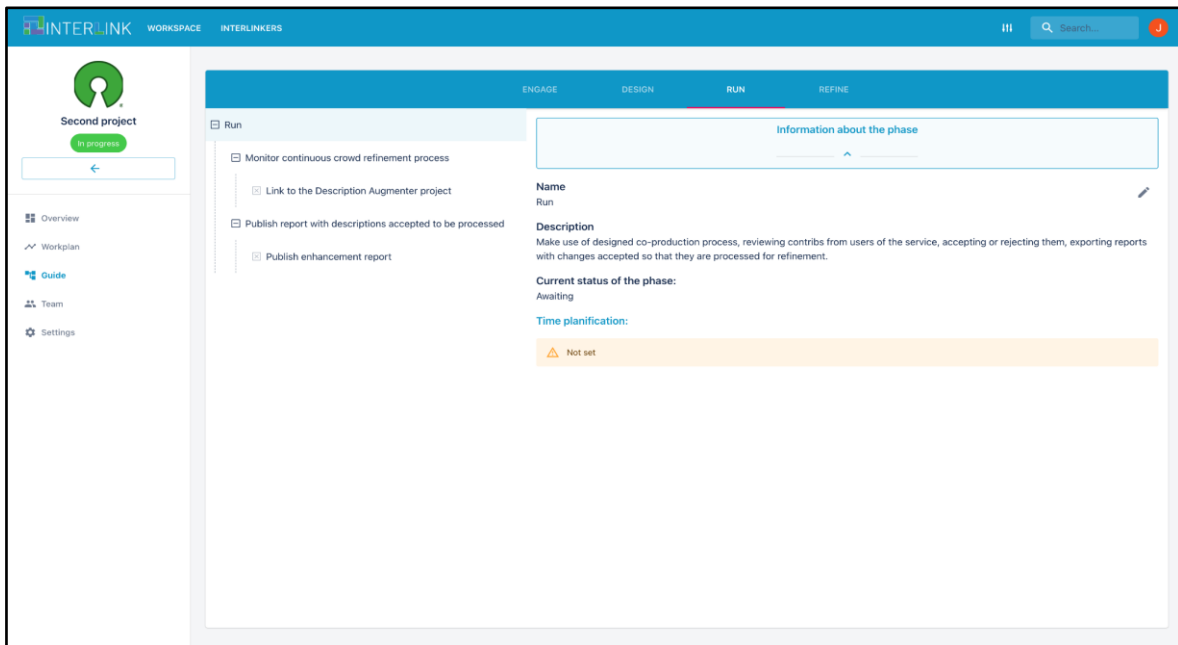


Figure 12. Comparison of INTERLINK BUILD (standard) vs. equivalent RUN (custom) stages in 2 different co-production projects.

3.3.1. INTERLINK Collaborative Environment Views

The Collaborative Environment offers different views to focus the co-production process in different aspects: a) guide; b) workplan and c) overview.

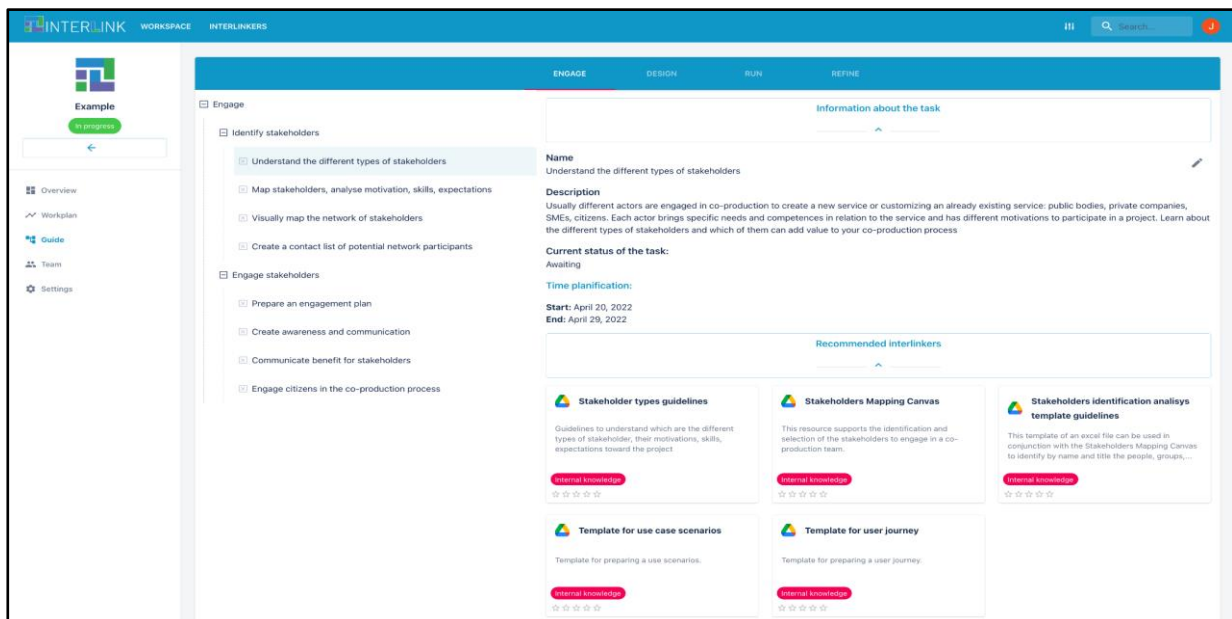


Figure 13. Guide section of the collaborative environment frontend.

The **guide view** showed in the Figure 13 shows how a co-production team can be guided in the co-production process, by being able to navigate through the co-production process phases, and for each phase, select a co-production objective, realise the tasks associated to that objective and get recommended relevant INTERLINKERS which may support accomplishing the objective of the currently selected task. Observe in the figure the selection of the task “Understand different types of stakeholders” with the corresponding INTERLINKERS being recommended, e.g. “Stakeholder types guidelines”, “Stakeholder Mapping Canvas” and so on.

Figure 14 shows the **Workplan view** which allows stakeholders to establish and review durations of the tasks accomplished within a co-production process. Figure 15 shows how the progress made in a co-production process can be reviewed easily by accessing the “**Overview**” view. Notice that navigation between a generated resource as result of having selected and used an INTERLINKER within a task is possible by means of the “See task” button. Also notice that navigation between the “Workplan” and “Guide” views is possible by clicking on the corresponding task name in the Workplan view (see Figure 14) or clicking on “Time planification” link within a given task view in “Guide” view (see Figure 13).

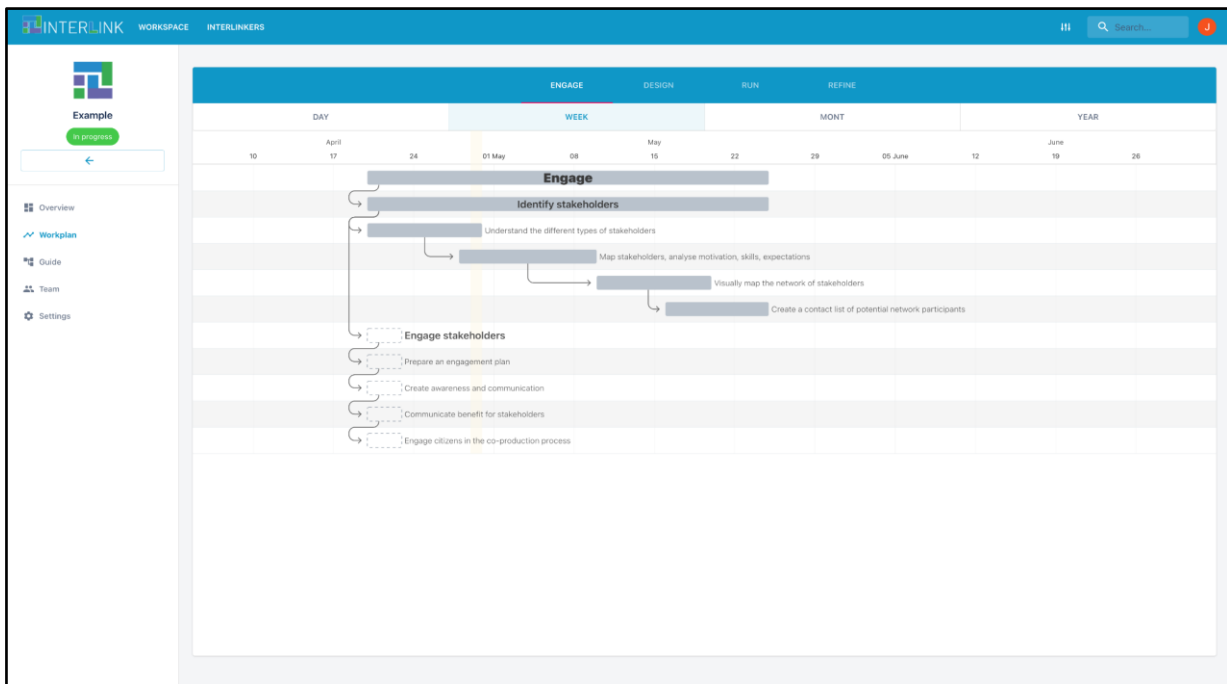


Figure 14. Workplan section of the collaborative environment frontend.

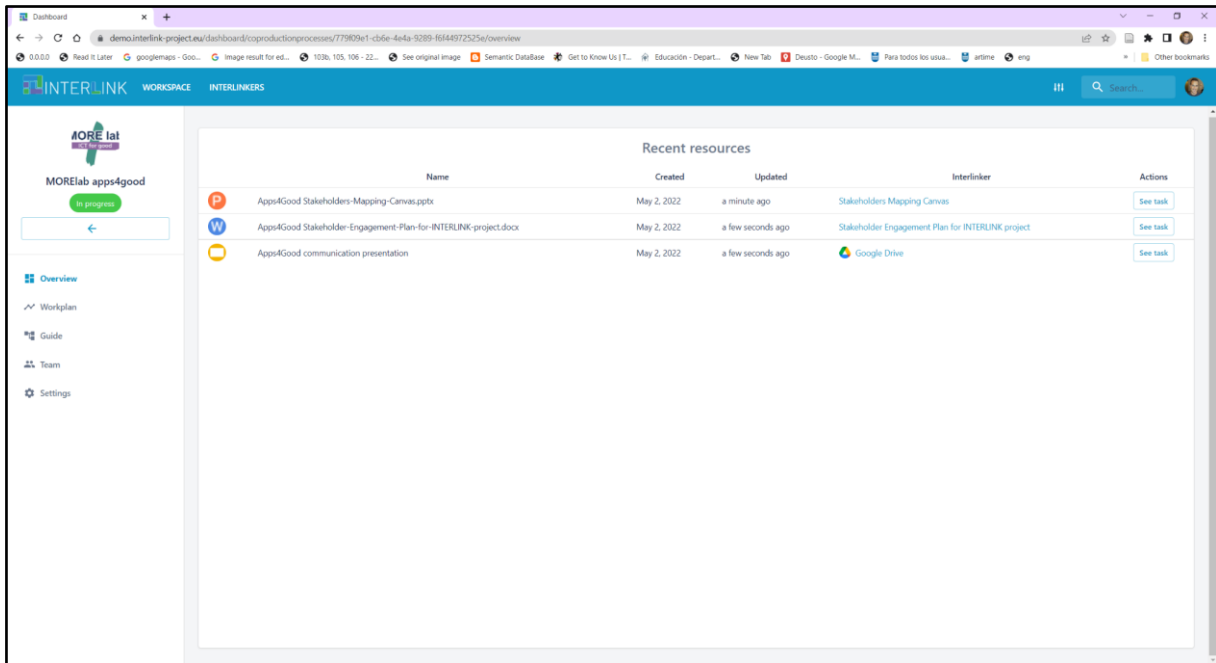


Figure 15. Overview view of the Collaborative Environment.

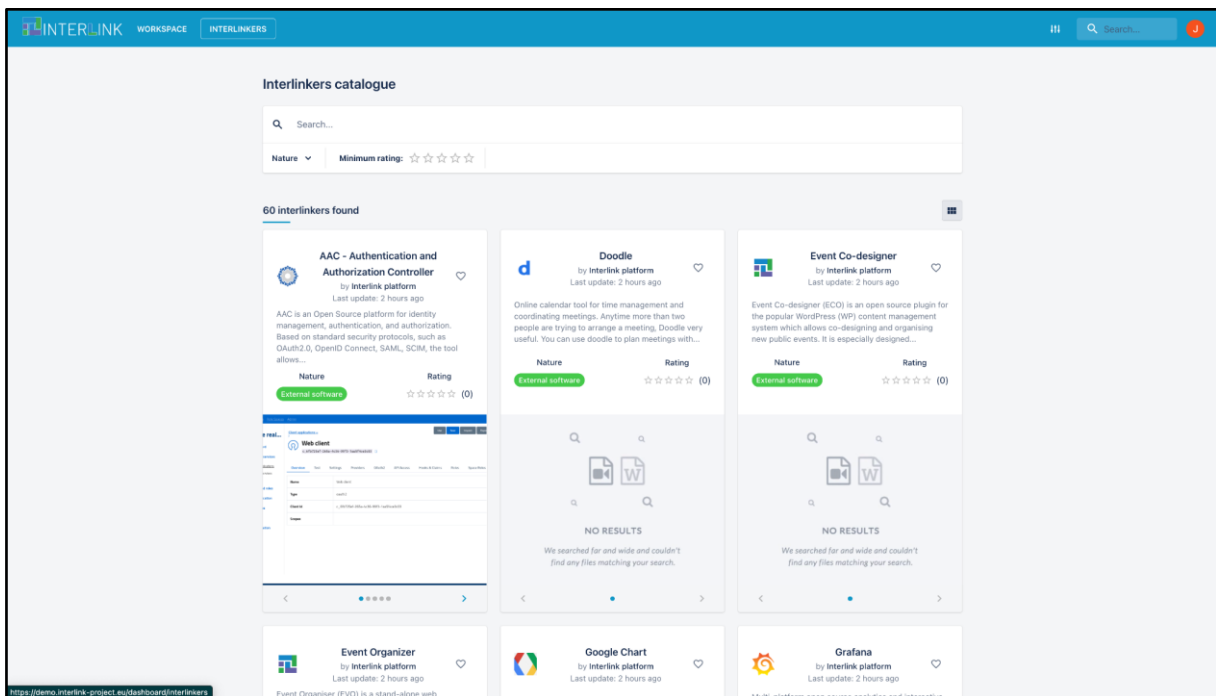


Figure 16. Interlinkers catalogue in the collaborative environment frontend.

These features will be provided through a web and mobile accessible responsive portal for all stakeholders involved in the INTERLINK community (PA, citizens, and private actors).



4 Preparation for the Pilot Cases for Evaluation

INTERLINK is going to be tested in 3 cross-European pilots. Firstly, in the Ministry of Economy and Finance - Italy (MEF) – a mock-up of a Participatory Strategic Planning Module (called PSPM) will be produced which allows Public Bodies and their staff to actively participate in the definition of the Strategic Plans, as well as to have access to a repository of good practices on strategic planning approaches and methodologies.

Secondly, at VARAM, the Ministry of Environmental Protection and Regional Development of the Republic of Latvia and its Latvian State Portal (<https://latvija.lv/EN>), which is a portal that provides easy access to services delivered by state and local government institutions. The goal is to continuously update and enhance such portal descriptions so that the public services published are increasingly adopted.

Thirdly, at Zaragoza city (ZGZ) and its Centre for Art and Technology (eTOPIA), where the aim is promoting collaborative city-making facilities and programs and improving the process of Open Innovation.

This task will provide the different INTERLINK instances, i.e. one per use-case site. Starting from the common ground of the pre-operational platform built in T4.3, this task will then be in charge of setting up and deploying a specific individual instance for each use-case. While the basis for all the instances is common, INTERLINK acknowledges the need for specific customisation when taken to the deployment and real use in the specific context of each use-case site.

This includes:

1. selection, integration and parameterization of INTERLINK enablers required for one particular instance;
2. fine tuning, according to the particularities of each local scenario, including (when/if necessary) small ad-hoc adaptations or bridges that could be needed, like e.g. the creation of parsers/gateway to integrate the local in-use data sources, systems, or legacy applications.

These stages and activities will be done in parallel for each use-case site (i.e. 3 subtasks for: Latvia, Spain and Italy).

As a result, an operational instance will be launched for each use-case, ready for evaluating INTERLINK on the 3 sites.

4.2 Guidelines for instantiation

Each environment uses a file containing certain environment variables that modify the behaviour and appearance of the platform components (.env.[environment-name] files).

In addition, volumes of data are used to mount certain digital resources, such as images, in the containers responsible for providing the platform's services. In this way, the



frontend is able to modify the images it displays depending on the environment where it is located.

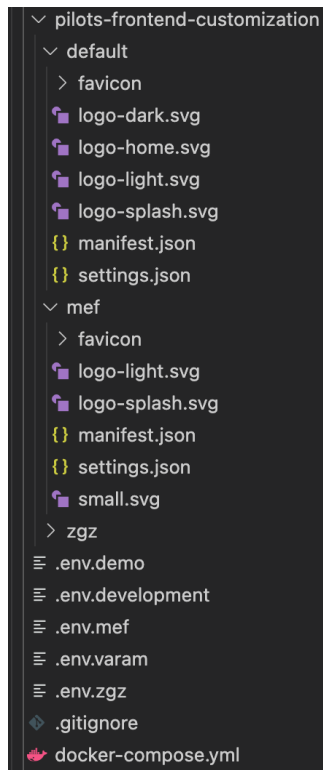


Figure 17. Interlinkers catalogue in the collaborative environment frontend.

Some logos and images can be modified in this way, as well as setting the default language and the allowed languages.

```
DOMAIN=dev.interlink-project.eu
MODE=production
PROTOCOL=https://

# pilot customization
FRONTEND_CUSTOMIZATION_IMAGES_PATH=./pilots-frontend-customization/default
PRIMARY_COLOR=
DEFAULT_LANGUAGE=en
ALLOWED_LANGUAGES=en,es,lv,it
```

Figure 18. Environment variables file for demo environment (.env.demo).

4.1.1. INTERLINKERS selection per Environment

As mentioned in the 3.1 section, each INTERLINKER is defined by a metadata.json file. This file contains the “environments” key, which defines in which environments must be launched.

```
{
  "name_translations": {
    "en": "Business Model Canvas"
  },
  "description_translations": {
    "en": "This canvas can be used collaboratively, for instance, during a brainstorming or a focus group, to reflect on the the most suitable business model associated to a co-produced service."
  },
  "environments": [
    "varam",
    "mef",
    "zgz"
  ],
  "languages": [
    "en"
  ],
  "problempfiles": [
    "SUS_PROBLEM_1"
  ],
  (...)
}
```

Figure 19. Reduced metadata.json file for Business Model Canvas knowledge INTERLINKER.

4.3 Specific Instantiations

4.1.1. Latvian Use-Case

The variables set for the Latvian use case set Latvian as the default language, allow the use of English, and point to the directory containing the logos and images to customise the frontend.

```
DOMAIN=varam.interlink-project.eu
(...)
# pilot customization
FRONTEND_CUSTOMIZATION_IMAGES_PATH=./pilots-frontend-customization/varam
PRIMARY_COLOR=
DEFAULT_LANGUAGE=lv
```



ALLOWED_LANGUAGES=en,lv

Figure 20. Reduced environment file for Latvian use case

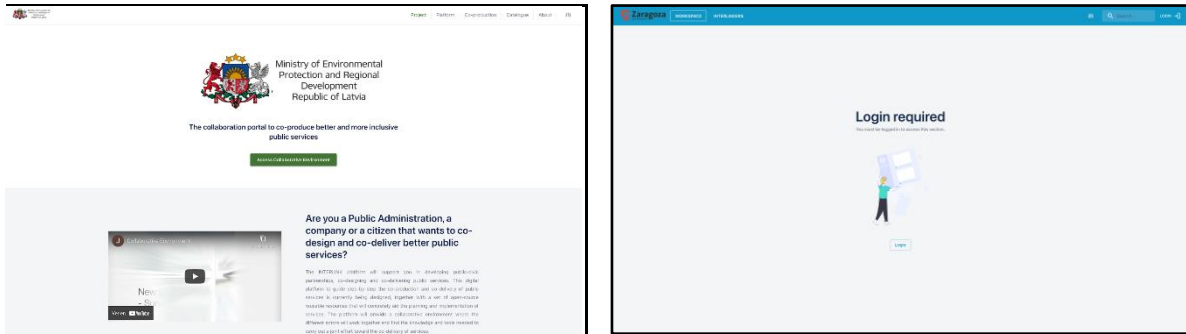


Figure 21. Result of the customization variables applied to the Spanish use case

4.1.2. Spanish Use-Case

The variables set for the Spanish use case set Spanish as the default language, allow the use of English, and point to the directory containing the logos and images to customise the frontend.

```
DOMAIN=zgz.interlink-project.eu
(...)
# pilot customization
FRONTEND_CUSTOMIZATION_IMAGES_PATH=./pilots-frontend-customization/zgz
PRIMARY_COLOR=
DEFAULT_LANGUAGE=es
ALLOWED_LANGUAGES=en, es
```

Figure 22. Reduced environment file for Spanish use case

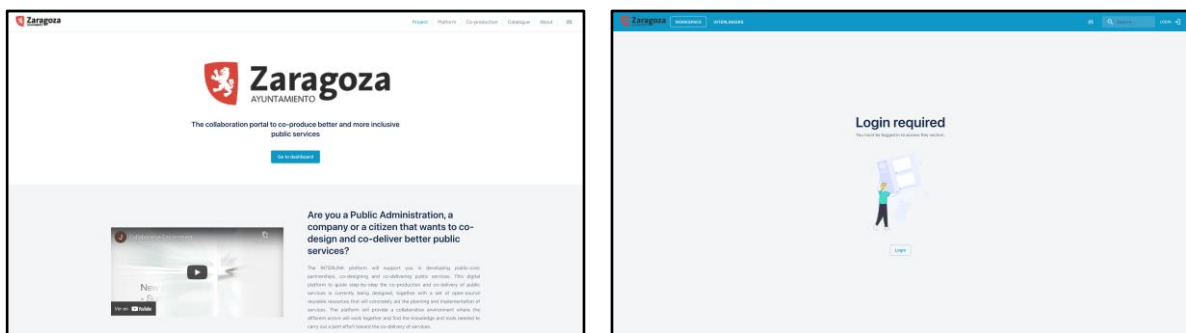


Figure 23. Result of the customization variables applied to the Spanish use case



4.1.3. Italian Use-Case

The variables set for the Italian use case set Italian as the default language, allow the use of English, and point to the directory containing the logos and images to customise the frontend.

```
DOMAIN=mef.interlink-project.eu
(...)
# pilot customization
FRONTEND_CUSTOMIZATION_IMAGES_PATH=./pilots-frontend-customization/mef
PRIMARY_COLOR=
DEFAULT_LANGUAGE=it
ALLOWED_LANGUAGES=en,it
```

Figure 24. Reduced environment file for Italian use case

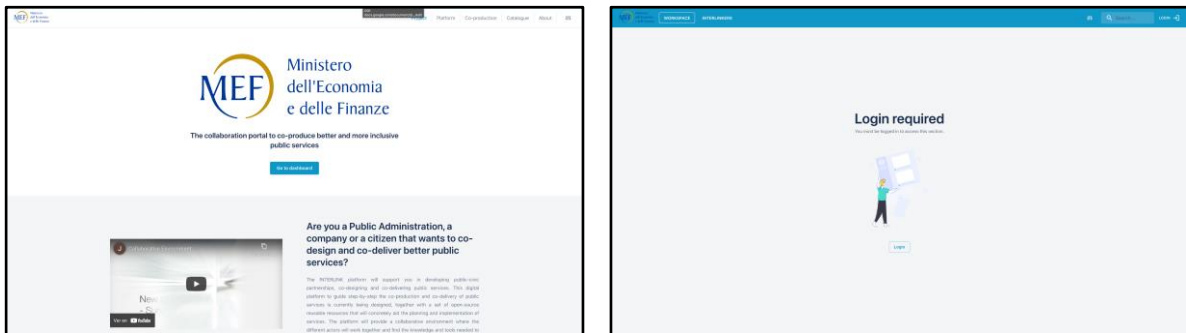


Figure 25. Result of the customization variables applied to the Italian use case