H2020 - INDUSTRIAL LEADERSHIP - Information and Communication Technologies (ICT)

ICT-38-2020: Artificial intelligence for manufacturing

Grant Agreement: 957362

# Explainable Manufacturing Artificial Intelligence

XMANAI

MAKING AI UNDERSTANDABLE

## WP2: Industrial Asset Management and Secure Asset Sharing Bundles

## D2.3: XMANAI Assets Management Bundles – Second Release

**Deliverable Leader:** FRAUNHOFER

**Due Date:** M30

**Dissemination Level:** Public

**Version:** 1.0

**Short Abstract**

The current deliverable provides a detailed technical documentation for the final release of the components that constitute the Asset Management Bundles Methods and thus, constitutes a report on the activities performed within all WP2 tasks. For each of the components the provided information includes the description of the implemented functionalities, documentation of the API, the technology stack that was employed for the development of the components and license information. Additionally, it provides the screenshots of the provided features through user interfaces. This deliverable can be used as a guide for the technical users and the researchers that participate in the implementation of the XMANAI platform or want to re-use its components and the business users that can be informed of the provided functionalities and how to utilize them.

**Further Information**: www.ai4manufacturing.eu

**Disclaimer**. The views represented in this document only reflect the views of the authors and not the views of the European Union. The European Union is not liable for any use that may be made of the information contained in this document. Furthermore, the information is provided “as is” and no guarantee or warranty is given that the information is fit for any particular purpose. The user of the information uses it at its sole risk and liability.

## Document Log

| | |
|---|---|
| Contributors | **FRAUNHOFER**, ATHENA, POLIMI, SUITE5, TXT, TYRIS, UBITECH |
| Internal Reviewer 1 | TXT |
| Internal Reviewer 2 | ATHENA |
| Type | Other |
| Delivery Date | 30/04/2022 |

## History

| Versions | Description |
|---|---|
| D0.1 | Initial ToC (Fraunhofer) |
| D0.2 | Contribution of partners to the assigned sections (All) |
| D0.3 | Alignment of the contributions, adding the section 1 and the Conclusion section (Fraunhofer) |
| R0.1 | Revision of internal reviewer 1 (TXT) |
| R0.2 | Revision of internal reviewer 2 (ATHENA) |
| D0.4 | Updated version addressing comments received during the internal review |
| F1.0 | Final version submitted to the EC |

# Executive Summary

The deliverable provides the documentation for the second and final release of the components of the XMANAI Asset Management Bundles, developed in WP2 "Industrial Asset Management and Secure Asset Sharing Bundles". It includes the following components: Data Storage Services, Data Handler, Provenance Engine, XAI Marketplace, Access Manager, Identity & Authorisation Management and Anonymiser.

For each component, the document provides information along the following dimensions: a) The description of the component functionalities and main features; b) The technical documentation of the provided APIs for each component; c) Screenshots of the core functionalities provided through the component's user interface where applicable; d) The technology stack and the implementation tools and libraries that were employed for the implementation; e) The licensing information for the produced software artefacts and the access details for the source code; f) The improvements provided in the final release of the component compare with the previous release presented in D2.2 "XMANAI Asset Management Bundles – First Release".

The main improvements provided in the second release of the XMANAI Asset Management Bundles and presented in this deliverable are the following:

- The components File Data Harvester and Provenance Engine, which were previously only planned, are now implemented and included in the second release of the bundles.
- A graphical user interface was implemented for the components API Data Harvester and Policy Editor.
- The version control feature, where different versions of an asset can be stored and CRUD operations can be performed on these versions, was added to the Assets Store. The Asset Store developed for the previous release didn't have this functionality. Additionally, the support of several types of assets was added and the list of possible queries was extended.
- The APIs and the underlying methods of most components were further extended to enable their integration with other XMANAI platform components and implementation of the planned workflows.

The second and final release of the components developed in WP2 "Industrial Asset Management and Secure Asset Sharing Bundles", which is documented in this report, will be integrated with the WP3 and WP4 components under the Beta release of the XMANAI platform that will be reported in deliverable D5.3 "XMANAI Platform – Beta Version" on M32. The further improvements of the components developed in the WP2 will be done as part of WP5 integration activities.

# Table of Contents

List of Figures

## List of Tables

# 1 Introduction

The main purpose of this section is to provide a brief overview of the deliverable, introducing the purpose of the document, its main content and possible dependencies with other XMANAI tasks and activities.

## 1.1 XMANAI Project Overview

Despite the indisputable benefits that Artificial Intelligence (AI) can bring in society and in any industrial activity, humans typically have little insight about AI itself and even less concerning the knowledge on how AI systems make decisions or predictions due to the so-called “black-box effect”. Many of the machine learning/deep learning algorithms are opaque and it is not possible to be examined after their execution to understand why a decision has been made. In this context, to increase trust in AI systems, XMANAI aims at rendering humans (especially business experts from the manufacturing domain) capable of fully understanding how decisions have been reached and what has influenced them.

Building on the latest AI advancements and technological breakthroughs, XMANAI focuses its research activities on Explainable AI (XAI) to make the AI models understandable and actionable at multiple layers (data-model-results). The project will deliver "glass box" AI models that are explainable to a "human-in-the-loop", without greatly sacrificing AI performance. With appropriate methods and techniques to overcome data scientists’ pains such as lifecycle management, security and trusted sharing of complex AI assets (including data and AI models), XMANAI provides the tools to navigate the AI’s “transparency paradox” and therefore:

(a) accelerates business adoption addressing the problematic that “if manufacturers do not understand why/how a decision/prediction is reached, they will not adopt or enforce it”, and
(b) fosters improved human/machine intelligence collaboration in manufacturing decision making, while ensuring regulatory compliance.

XMANAI aims to design, develop and deploy a **novel Explainable AI Platform** powered by explainable AI models that inspire trust, augment human cognition and solve concrete manufacturing problems with value-based explanations. Adopting the mentality that “AI systems should think like humans, act like humans, think rationally, and act rationally”, a catalogue of **hybrid and graph AI models** is built, fine-tuned and validated in XMANAI at 2 levels: (i) baseline AI models that will be reusable to address any manufacturing problem, and (ii) trained AI models that have been fine-tuned for the different problems that the XMANAI demonstrators’ target. A bundle of **innovative manufacturing applications and services** are also built on top of the XMANAI Explainable AI Platform, leveraging the XMANAI catalogue of baseline and trained AI models.

XMANAI will validate its AI platform, its catalogue of hybrid and graph AI models and its manufacturing apps in **4 realistic, exemplary manufacturing demonstrators** with high impact in: (a) optimizing performance and manufacturing products’ and processes’ quality, (b) accurately forecasting product demand, (c) production optimization and predictive maintenance, and (d) enabling agile planning processes. Through a scalable approach towards Explainable and Trustful AI as dictated and supported in XMANAI, manufacturers will be able to develop a robust AI capability that is less artificial and more intelligent at human and corporate levels in a win-win manner.

## 1.2 Deliverable Purpose and Scope

The current deliverable D2.3 “XMANAI Asset Management Bundles – Second Release” presents the final release of the XMANAI asset management components developed within all WP2 tasks. It updates the previous deliverable D2.2 “XMANAI Asset Management Bundles – First Release”. The document provides a supportive documentation for the final release of the components that constitute the XMANAI Asset Management Bundles: Assets Store, Provenance Engine, API Data

Harvester, File Data Harvester, File/Data Manager, Data Exporter, Registry/Metadata Manager, Contract Manager, Policy Engine, Policy Editor, Identity & Authorisation Management and Anonymiser.

The implemented functionalities of the aforementioned components in the current release are explained in this document, as well as the main improvements for all developed components compared to the previous version. The deliverable also provides detailed information for each XMANAI Asset Management Bundle, including the architecture and the technology stack employed during the development of the components, the user interfaces of the components, as well as licensing and access information. Finally, assumptions made for the implementation of the current release and restrictions and challenges identified during the WP2 activities are presented and discussed in this report.

The components presented in this deliverable will be integrated in the second version of the XMANAI Platform (D5.3 "XMANAI Platform - Beta Version") in the context of the WP5 activities.

## 1.3 Impact and Target Audiences

As in the previous deliverables of the work package WP2, this document targets mainly the technical users that develop the XMANAI Platform, as well as the researchers who support the solution. The business users also belong to the target audience, as this report describes the components and their provided functionalities and present snapshots of user interfaces that explain how these components can be used.

Section 2 contains information regarding the available functionalities of the XMANAI Asset Management bundles, the technology used, the provided interfaces as well as the improvements since the previous version. This knowledge is of use for all stakeholders as some can use this report to help them understand what is available and how it can be used, and others understand and identify limitations and proceed to improve them.

## 1.4 Deliverable Methodology

The information reported in this deliverable has been produced by the consortium members following the methodology described below:

Following the initial definition of the WP2 architecture and the components it comprises, the XMANAI partners proceeded with the implementation of the designed functionalities according to the project's development work plan. The partner leading the implementation of each component was responsible for providing the technical information in this report that documents the implementation activities carried out towards this final release.

## 1.5 Dependencies in XMANAI and Supporting Documents

This document provides a supportive report for the final release of the XMANAI Asset Management Bundles. The components and their implemented functionalities described in this document, refer to the expected functionalities that were introduced and discussed in D2.1 "Asset Management Bundles Methods and System Designs" while their first version was presented in the previous deliverable D2.2 "XMANAI Asset Management Bundles – First Release". The document provides information about new features, modifications, and enhancements according to the already planned functionalities and received feedback.

## 1.6 Document Structure

This document is structured as follows:

- Section 2 provides information regarding the implementation details of the first release of the XMANAI Asset Management Bundles covered in this document, that include technologies and tools, internal design and architecture and licensing information. Moreover, the current implementation status is presented, as well as the planned functionalities for the next release of the XMANAI Asset Management Bundles.
- Section 3 concludes the document and summarises the next steps.

# 2 XMANAI Asset Management Bundles

## 2.1 Introduction

The XMANAI Asset Management Bundles provide the assets management and sharing as well as identity and authorisation management functionalities for the overall XMANAI platform. The figure below presents the conceptual architecture of the final release of the XMANAI Asset Management Bundles.

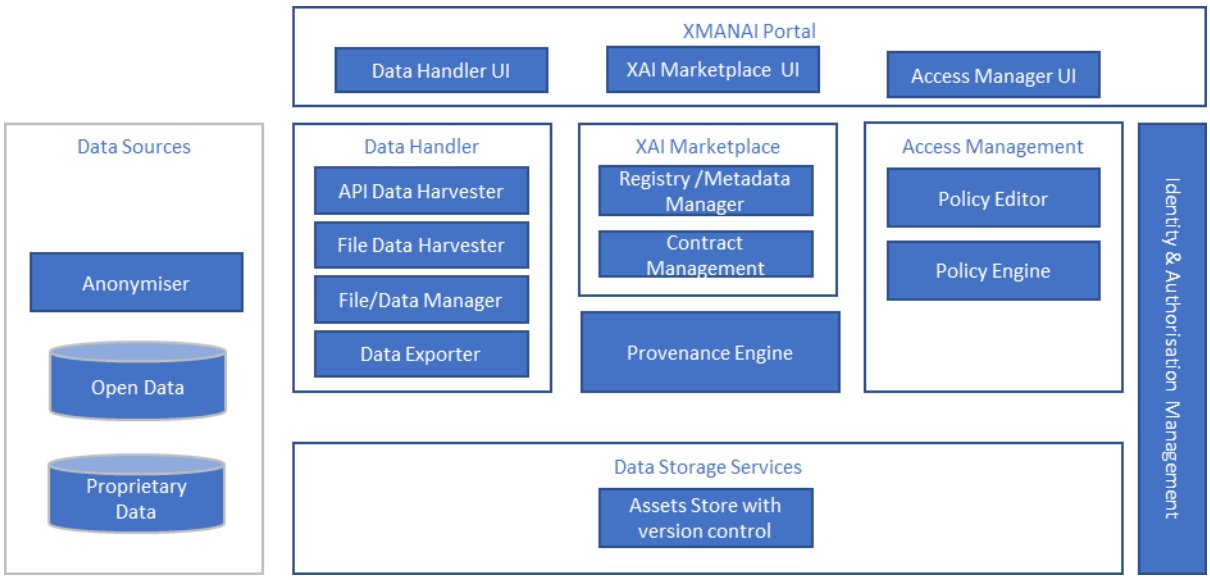


*Figure 2-1: Overview of components providing industrial assets management and sharing functionalities*

The final version of the architecture remained almost unchanged compared to the previous version documented in the D2.2. In the new version the Provenance Engine and its storage are shown in the diagram and described in the document as one component.

The XMANAI assets are stored, managed and versioned by the Assets Store with the version control playing the role of the Data Storage Service in XMANAI. It provides an API to get and push data and other assets from/to it. The API calls trigger the logging of modifications and access to assets with help of the Provenance Engine.

The Data Handler enables data collection from external data sources to the Platform Storage Service, data export from XMANAI and access to the XMANAI data for external 3rd party services through an API. The Data Handler consists of the:

- API Data Harvester, responsible for collecting data from external APIs.
- File Data Harvester, responsible for transforming data files to the XMANAI data model, saving data as XMANAI datasets in the storage and recording the datasets metadata in the Registry/Metadata Manager. All XMANAI datasets are conformed to the XMANAI data model data accompanied by the describing metadata. Only datasets are utilized in the pipeline of XMANAI.
- Data Exporter, responsible for exporting data snapshots in requested file formats.
- File/Data Manager, responsible for managing the files (upload, delete, edit) that will be used in the forthcoming ML steps.

The Data Anonymizer is realised as a standalone tool executed on the data provider side. The data will be anonymised before being shared with the XMANAI platform. The security mechanisms in the platform are realised with help of Identity and Access Manager, Policy Engine and Policy Editor. Data

sharing between the data provider and data consumers is implemented in the XAI Marketplace component, consisting of the Contract Manager and Registry/Metadata Manager, whose mission in the platform is the management of metadata for all data, analytical models and other assets in XMANAI and the controlled, contract-based sharing of assets among stakeholders.

The following sections present the implemented components in detail.

## 2.2 Data Storage Services: Assets Store with Version Control

### 2.2.1 Description

The Assets Store with Version Control is a centralized database system for the XMANAI platform that stores all XMANAI assets. All assets are stored either as a relational table or as a binary file. The two types of assets stored in the Assets Store are:

- Structured Data conforming to the XMANAI data model. These include assets such as "Datasets" or "Results" in the form of relational tables, where the data schema of the table conforms to the XMANAI data model.
- Binary files, that include trained models, files, script files or any other files that are needed as part of the XMANAI pipeline.

The assets are treated differently in the Assets Store based on their type. Currently, the asset types available to the XMANAI users are, *Dataset, Result, File, Model, Scripts, Explanation, Detector, MG_Model* and *Pipeline*. The Assets Store offers an API that has endpoints to perform all CRUD operations on the stored assets. It can be used to create, read, update, and delete relational tables and binary files. The version control feature of the Assets Store helps to avoid losing data or other important information in the experimental XMANAI multi-user data analytics environment. Several versions of an asset can be stored and retrieved using this feature. The relational tables and the binary files are stored in two separate databases which are connected to the REST API endpoints. These endpoints can be used to create new assets or new versions of an existing asset. The databases used to store the assets are PostgreSQL.

Some important features of the Assets store are:

- Similar to all XMANAI components, the API requests of the Assets Store have to be authorized using a Bearer token or a Cookie.
- All assets stored in the Assets Store have a UUID (Universal Unique Identifier) which is generated by the Assets Store when an asset is created. The UUID, the name of the asset and any other metadata provided during the creation of an asset, are stored in the Registry/Metadata Manager. All versions of an asset have a common metadata entity stored in the Registry, and any change in the metadata will result in the creation of a new asset.
- When an asset is created in the Assets Store, the access policy of the asset is set to CONFIDENTIAL in the XMANAI Policy Manager. The access policies of assets can be edited later through the Policy Manager. The access policy is only created once at the creation of a new asset and is not changed when an asset gets updated.
- Similar to the creation of newer versions of an asset, update operations on assets will not result in any changes to the metadata or access policy of an asset. Deletion of an asset will result in removing the access policy of an asset, but the metadata is retained in the Registry.
- Each version of an asset is associated with a version id in the form *v1.0, v1.1* etc. A list of version IDs associated with an asset is stored in the Registry and every time a new version of an asset is created, this list of version IDs will be updated in the Registry.
- For assets stored under the asset type "Result", different versions can be allowed to have different data models and this will be mapped and stored in the metadata in the Registry.

- Tabular assets can be retrieved as a whole or as individual rows in sorted formats using query parameters such as DISTINCT, OFFSET and LIMIT. Summary statistics and count of occurrences of values are also available.
- Along with the CRUD operations on binary files, it is also possible to rename files and to retrieve a list of all file names, UUIDs, file extension types and its version ids.
- Along with the assets stored in the Assets Store, API Data Harvester requires two types of additional assets which are JSON objects called *harvesting pipes* and JavaScript files called *transformation scripts.* These are also assets that belong to the asset type "Scripts" but cannot be stored in the Assets Store as the API Data Harvester cannot access external databases to retrieve these files. These files are stored in a Gitlab repository and since these files should have access policies, access to these files is provided through API endpoints in the Assets Store.

*Table 2-1: Assets Store – Implemented Functionalities*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| AS.1 | Assets Store for Binary Files<br><br>This functionality allows the user and other XMANAI components to create, read, delete and update binary files in the Assets Store. The database used to store binary files is PostgreSQL and if some heavy memory is allowed, binary files up to a size of 1GB can be stored in a column of type *byte.* Files belonging to each asset type has a UUID and is stored separately. |
| AS.2 | Assets Store for Non-Binary Files/Relational Tables<br><br>This functionality allows the user and other XMANAI components to create and update non- binary files or relational tables that conform to the XMANAI data model in the Assets Store. These files include structured data in the form of tables aligned with the XMANAI data model. Each of the assets are stored as tables and the name of the table is replaced with a UUID. This UUID is provided to the user as a response upon successful creation of an asset. It is also stored in the Registry along with the actual name of the table and any other metadata that is available. |
| AS.3 | Version Control for Assets Store<br><br>The version control feature of the Assets Store identifies and stores separate versions of an asset. Every time new data is harvested using the Data Handler, they can be stored as a new version of an existing asset or as a new asset depending on If there are any metadata model changes. Every version of an asset is identified using a version ID, which is in form *v1.0, v1.1* etc. These version IDs are used by the Provenance Engine to track changes made by a user to an asset. Along with allowing CRUD operations on separate assets, it's also available for separate versions of an asset. |
| AS.4 | Access to API Data Harvester scripts<br><br>Transformation scripts for the API Data Harvester data transformation processes are being stored in the Assets Store as well. These files belong to the asset type "Result" and should have the necessary access policies set and metadata stored in the Registry. The frontend of the API Data Harvester stores them through a dedicated Assets Store endpoint. |

### 2.2.2 API documentation

A brief overview of the API endpoints is available in the following table and the swagger file for the documentation is present in the GitLab repository[1].

*Table 2-2: Assets Store – API Endpoints Overview*

| Endpoint | Method | Description |
|---|---|---|
| Tabular Assets Store (Asset type: Dataset, Result) | | |
| /api/assets/create_table | POST | Add new asset tables or create new versions of an existing table. |
| /api/assets/get_tables | POST | Retrieve multiple asset tables. |
| /api/assets/get_fields | POST | Retrieve certain fields from an asset table. |
| /api/assets/count_values | POST | Count the occurrences of values within a single column. |
| /api/assets/summary_statistics | POST | Calculate the summary statistics (min, Q1, Q2, Q3, max). |
| /api/assets/update_version | PUT | Update particular versions of an asset table. |
| /api/assets/delete_version | DELETE | Delete one version of an asset from the assets store. |
| /api/assets/delete_tables | DELETE | Delete several asset tables. |
| File Assets Store (Asset type: Dataset, Result, Explanation, MG_Model, Detector, File, Model, Pipeline, Script) | | |
| /api/assets/create_file | POST | Add a new binary file into the Assets Store. |
| /api/assets/get_file | GET | Get a binary file from the Assets Store. |
| /api/assets/rename_file | PUT | Rename a binary file in the Assets Store. |
| /api/assets/delete_file | DELETE | Delete an existing binary file from the Assets Store. |
| /api/assets/get_all_names | GET | Get all existing filenames, UUIDs, filetypes and version_ids of binary files. |

### 2.2.3 User Interface

The Assets Store with Version Control is a back-end component with a PostgreSQL database and a REST API for interacting with the database and rest of the XMANAI components. It does not have a GUI interface.

### 2.2.4 Technology stack and License information

[1] https://GitLab.com/xmanai-h2020/api-docs/-/blob/main/Assets Store with Version Control.yaml

The Assets Store with Version Control has a PostgreSQL database with a REST API attached to it to perform all data access operations. The API is written in Python using the Flask framework. All parts of this component are published under Apache-2.0-license.

### 2.2.5 Code repository

The code of the component is published in a public section of the XMANAI repository in GitLab[2].

### 2.2.6 Improvements since the previous version

Since the previous release several changes and improvements were made, which are listed as follows:

- Along with basic CRUD operations on a tabular asset, additional endpoints to retrieve only some rows of an asset using query parameters such as OFFSET, LIMIT, and DISTINCT are available. Along with this, the rows can also be retrieved in a sorted manner.
- To support the developed in WP3 component Results and Visualization Engine, endpoints to count the occurrences of a value and summary statistics are added.
- To support the API Data Harvester, *harvesting pipes* and *transformation scripts* can be created, retrieved and updated using the Assets Store endpoints.
- To support the File Data Harvester and File Data Manager, a list of all file names, file extension types, UUIDs and version ids are added.
- Assets Store with Version Control is also responsible to create the access policy in the Policy Manager and to store initial metadata of an asset in the Registry when an asset creation is successful.
- The version control feature, where different versions of an asset can be stored and CRUD operations can be performed on these versions, is available. Version ids are added to each version of an asset, which is used to identify the different versions.

### 2.2.7 Considerations

At the time of writing, no foreseen challenges or limitations are present for proceeding with further development of the Assets Store with Version Control.

## 2.3 Data Handler: API Data Harvester

### 2.3.1 Description

The API Data Harvester (ADH) is a service responsible to import (i.e. download) data and metadata from endpoints on the web according the predefined time plan. These endpoints can come in all shapes and forms, for example simple public REST APIs, restricted SQL dumps or geodata streams. Thus, ADH constitutes a solution for an automated data ingestion procedure offering a pipeline management system, which allows for an individualized and schedulable transformation and transport of large amounts of data from external APIs into the Assets Store. It therefore represents a bridge connecting external data sources to the Assets Store while ensuring that the data format aligns with the requirements. It consists of five individual components, which communicate via HTTP or TCP/IP:

- The **Scheduler** is the component, through which a harvesting process is triggered. When addressed, it expects the type of **Trigger** (daily, immediate, etc.) to be passed as an argument inside the request body as well as well as the name of the respective **Pipe Object** to be

[2] https://GitLab.com/xmanai-h2020

specified in the endpoint URL. The **Pipe Object** is a JSON-file, which contains all relevant information in regards to the harvesting process, such as the URL of the data source. Hence, one **Pipe Object** corresponds to one specific harvesting process and wanting to harvest a new data source requires the creation of a new **Pipe Object**. They can be stored either locally or in a Gitlab repository and then be accessed by the **Scheduler** when triggered. **Triggers** can be set either in a manual and immediate manner or they can be set once in a way to initialize an indefinite number of harvesting processes in a reoccurring manner (e.g., every 24h). When triggered, the **Scheduler** will retrieve the **Pipe Object** and forward it to the **Importer**.

- The **Importer** receives the **Pipe Object**, queries the endpoint specified in it, writes the yielded data into it and forwards it to the **Transformer**.
- The **Transformer** extracts the data out of the **Pipe Object** and transforms it utilizing the transformation script, which is also mentioned in the **Pipe Object**. The transformation process gives the opportunity to specify the data model of the asset, also to add metadata. The transformed data and metadata are being written into the **Pipe Object** and passed to the **Exporter**.
- The **Exporter** forwards the data to the Asset Store in order to create a new asset. The UUID of the just created asset is being stored along with the UUID of the **Pipe Object**. This way, a connection between the harvesting process and the resulting asset is being established in order to allow for an easy update process, instead of having to create a new asset with every data pull.

*Table 2-6: API Data Harvester – Functionalities and Status*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| ADH.1 | Creation and Management of **Pipe Objects**:<br>• Creating Pipe Objects and offering a possibility to manage them |
| ADH.2 | Use case specific Configuration of the Data Harvesting Parameters through **Pipe Objects** (implemented):<br>• Selecting individual API endpoints for data harvesting<br>• Configuring individual data-transformation/-cleansing/-anonymization<br>• Configuring metadata and the data model |
| ADH.3 | Time specific execution of Harvesting Processes through **Triggers** (implemented):<br>• Either manual and immediate or automated and interwall-wise triggering of Harvesting Processes<br>• Creating a new asset<br>• Updating an existing asset |
| ADH.4 | A GUI, serving all of the above-mentioned functionalities to the user in an intuitive manner (in progress):<br>• Guided step-by-step pipeline creation process including data transformation<br>• Menu for Individual trigger scheduling<br>• Pipeline Monitoring with detailed live information<br>• Overview of all active and inactive pipelines |

| | |
|---|---|
| | • Menu for pipeline editing/deleting |

### 2.3.2 API documentation

For the **Scheduler** the API documentation is to be found in the project GitLab repository[3]. The table below provides a brief overview:

*Table 2-3: API Data Harvester – API Endpoints Overview*

| Endpoint | Method | Description |
|---|---|---|
| /pipes | GET | Get a list of all **Pipe Objects**. |
| /pipes/{pipeName} | GET | Get specified Pipe **Object**. |
| /pipes/{pipeName}/triggers | GET | Get list of **Triggers** associated with specified **Pipe Object**. |
| | DELETE | Delete all **Triggers** associated with specified **Pipe Object.** |
| /pipes/{pipeName}/triggers/{triggerId} | GET | Get specified **Trigger** of specified **Pipe Object**. |
| | PUT | Create or update specified **Trigger** of specified **Pipe Object**. |
| | PATCH | Merge content with specified **Trigger** of specified **Pipe Object**. |
| | DELETE | Delete specified **Trigger** of specified **Pipe Object**. |
| /triggers | GET | Get a list of all **Triggers**. |
| | PUT | Bulk update all **Triggers**. |

[3] https://GitLab.com/xmanai-h2020/api-docs/-/blob/main/API Data Harvester Scheduler.yaml

The **Pipe Object** constitutes a data object rather than an API but it still is a crucial element since it defines the whole harvesting process. It can be found in the project repository[4]. The table below provides a short overview regarding the most important attributes of this data object:

*Table 2-4: API Data Harvester – Pipe Object Attributes*

| Component | Attribute | Description |
|---|---|---|
| Header | name | Unique name of the **Pipe Object** |
| Importer (Segment 1) | body/config/address<br><br>body/config/inputFormat | API-endpoint to harvest the data from<br>Format of the data to be harvested |
| Transformer (Segment 2) | body/config/script | Contains the transformation script |
| Exporter (Segment 3) | body/config/pipeID | Contains the automatically created pipeID, which will be associated with the asset, which is being created during the harvesting process. |

### 2.3.3 User Interface

The figures below give an overview over the GUI for the most important ADH functionalities.

[4] https://GitLab.com/xmanai-h2020/api-docs/-/blob/main/API Data Harvester Pipe Object.json

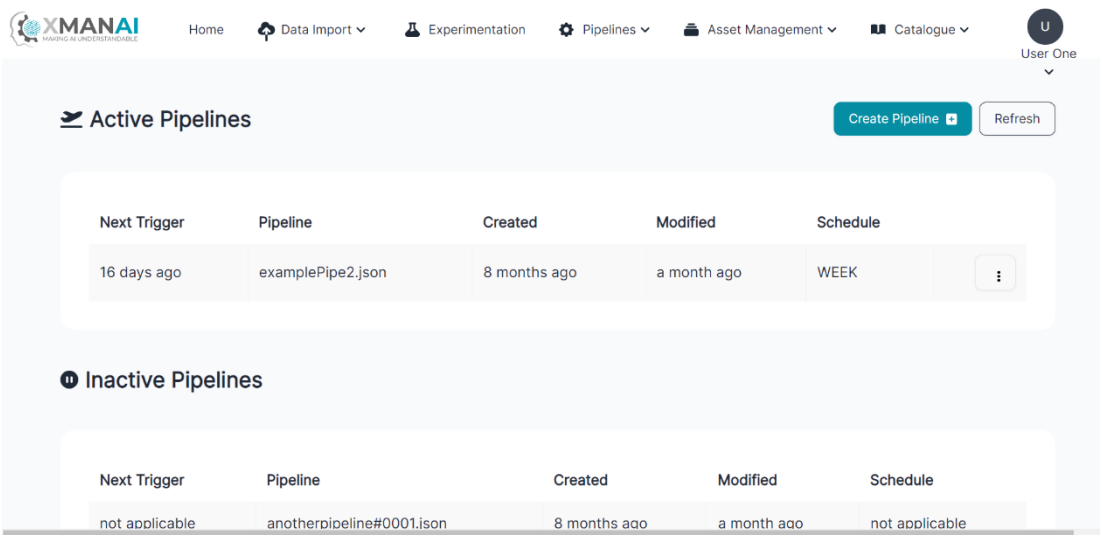


*Figure 2-2: ADH - Overview of active and inactive pipelines*

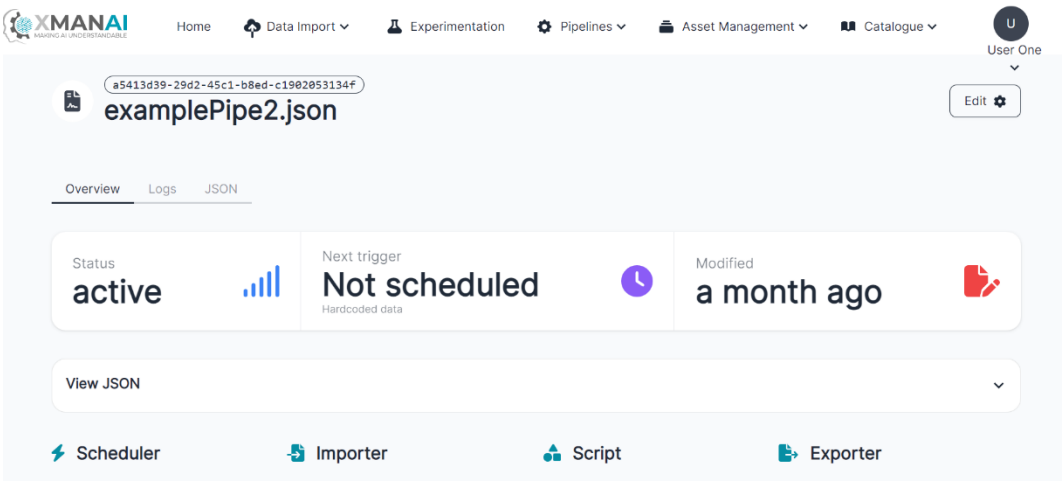


*Figure 2-3: ADH - Monitoring view of an active pipeline*

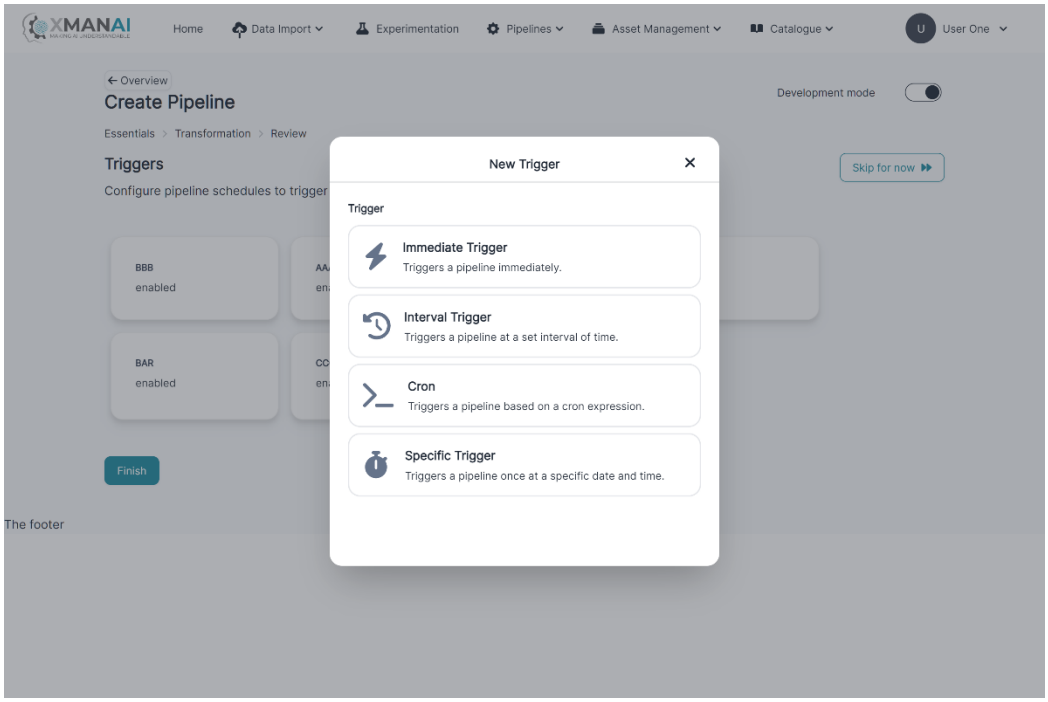


*Figure 2-4: ADH - Scheduler configuration menu*

### 2.3.4 Technology stack and License information

The **Importer** is written in Kotlin, the other components, namely **Scheduler**, **Transformer** and **Exporter** are written in Java. The API of each of the components, through which the communication is handled, is provided by the **Pipe Connector Module**. Thus, it is written in Kotlin, works asynchronously in an event-based manner, utilizing the Vert.x-Library[5] and allows several thousand requests to be handled simultaneously. The frontend is implemented in Vue.js[6].

### 2.3.5 Code repository

The code of the component is published in a private section of the XMANAI repository in GitLab[7], access to which can be provided by a request sent to the project coordinator.

### 2.3.6 Improvements since the previous version

Since the previous release, the ADH has been integrated with the Identity and Authorization Manager and is able to handle cookie-based authorization. Additionally, a frontend, which previously was not present at all, is implemented for this release.

### 2.3.7 Considerations

At the time of writing, no foreseen challenges or limitations are present for proceeding with further development of the API Data Harvester.

[5] https://github.com/vert-x3/vertx-lang-kotlin
[6] https://vuejs.org
[7] https://GitLab.com/xmanai-h2020

## 2.4 Data Handler: File Data Harvester

### 2.4.1 Description

The File Data Harvester (FDH) provides an alternative way for ingesting data into the Assets Store. Instead of harvesting through external APIs, as done by the ADH, data can also be extracted from files, which are already present in the Assets Store. It thus constitutes an internal conversion module, enabling the extraction and transformation of data out of non-conformed XMANAI data model sources to the XMANAI data structured format.

Similar to the ADH, the FDH utilizes a GUI in order to guide the user through the harvesting process. In a first step, the user is prompted to select one file through the File Browser. Subsequently, they have to transform the data from its original format to a model as foreseen by the XMANAI data model. A drop-down menu with a fixed domain of selectable attributes will help them with that. As a last step, they must give the asset to be stored a name and can as well assign a description to it, before completing the process and storing it inside the Assets Store. While the original file will be kept, the user now also has its data available in a structured format, enabling them to utilize it for further processes (e.g., using it as training data for ML purposes).

*Table 2-5: File Data Harvester - Functionalities and Status*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| FDH.1 | File Browser<br>Giving the user an overview of all binary files, which are stored in the Assets Store and associated to him so that he can select one of them. |
| FDH.2 | Data Preview and Conversion<br>Giving the user a visual representation of the data stored inside the file, which he selected and guiding him through the interactive process of converting the data model to the form required by the XMANAI eco system. |
| FDH.3 | Ability to provide the basic metadata for the dataset<br>Letting the user add metadata to the converted asset before storing it in the Assets Store. |
| FDH.4 | Storing the converted asset inside the Assets Store<br>This is based on the information provided by the user about the structure and semantics of the data. This feature will not be provided as part of this release. |
| FDH.5 | Support multiple file types<br>Support the harvesting of binary files in csv-, txt- and xls-format. |

### 2.4.2 API documentation

A brief overview of the endpoints of the FDH backend can be found in the below table. Additionally, it is present in this repository[8]. Note that this API is solely to be utilized by the FDH frontend.

[8] https://gitlab.com/xmanai-h2020/api-docs/-/blob/main/File data harvester.yaml

*Table 2-6: File Data Harvester – API Endpoints Overview*

| Endpoint | Method | Description |
|---|---|---|
| /get_asset | GET | Retrieve a specified binary asset from the Assets Store.<br>Arguments:<br>• file_type<br>• uuid |
| /get_filenames | GET | Get a list of all names associated with binary files inside the Assets Store. |
| /livesearch_data_model | POST | Receive a list of all XMANAI data model properties corresponding to the input string.<br>Arguments:<br>• sequence |
| /save_asset | POST | Store the converted asset inside the Assets Store.<br>Arguments:<br>• uuid |

### 2.4.3 User Interface

As mentioned above the FDH offers a GUI. Below, figures of the most important frontend features can be found.

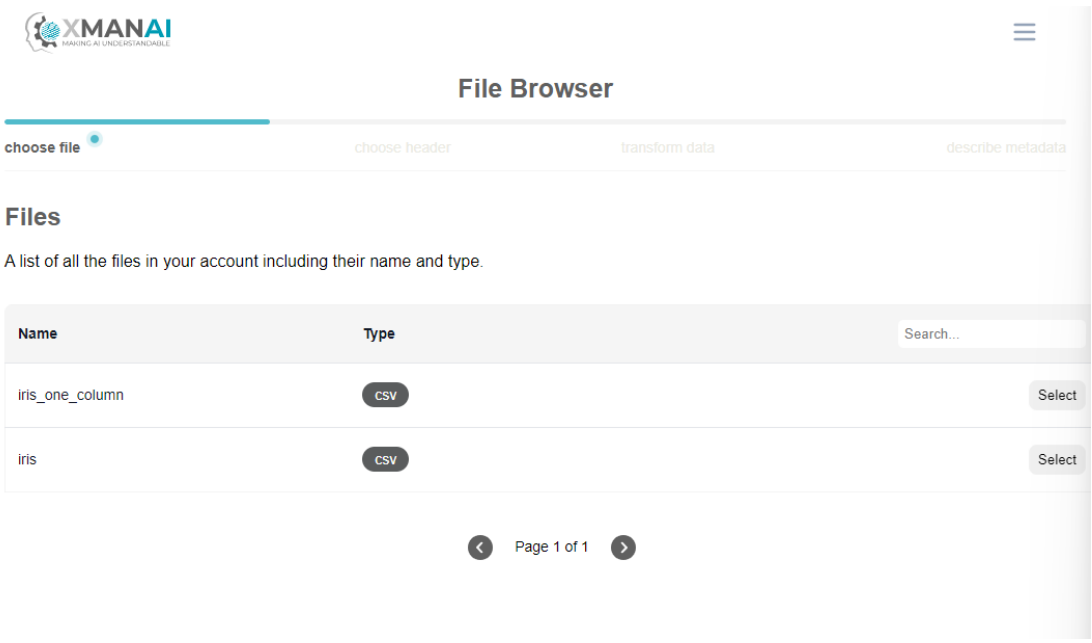


*Figure 2-5: FDH - File Browser with pagination and search function*

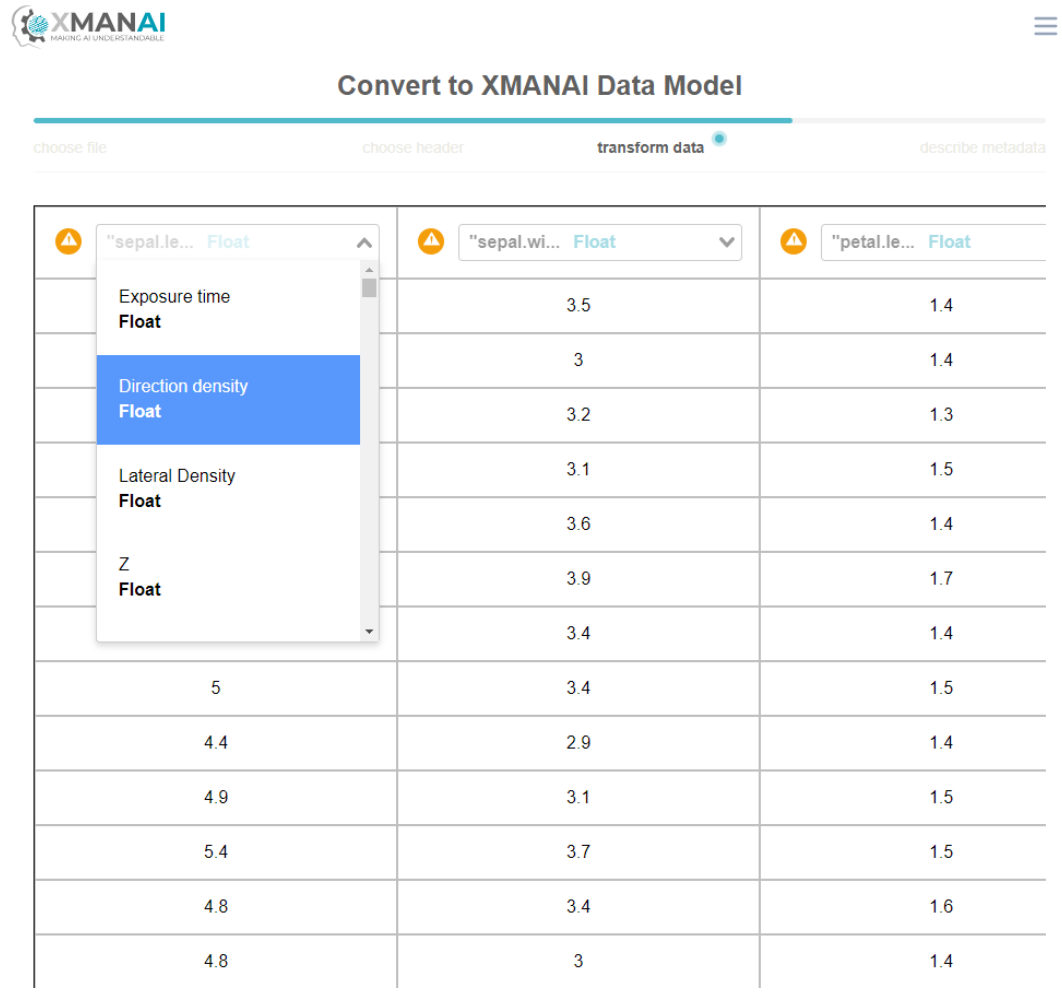


*Figure 2-6: FDH - Data Conversion Tool with drop-down menu and integrated search function*

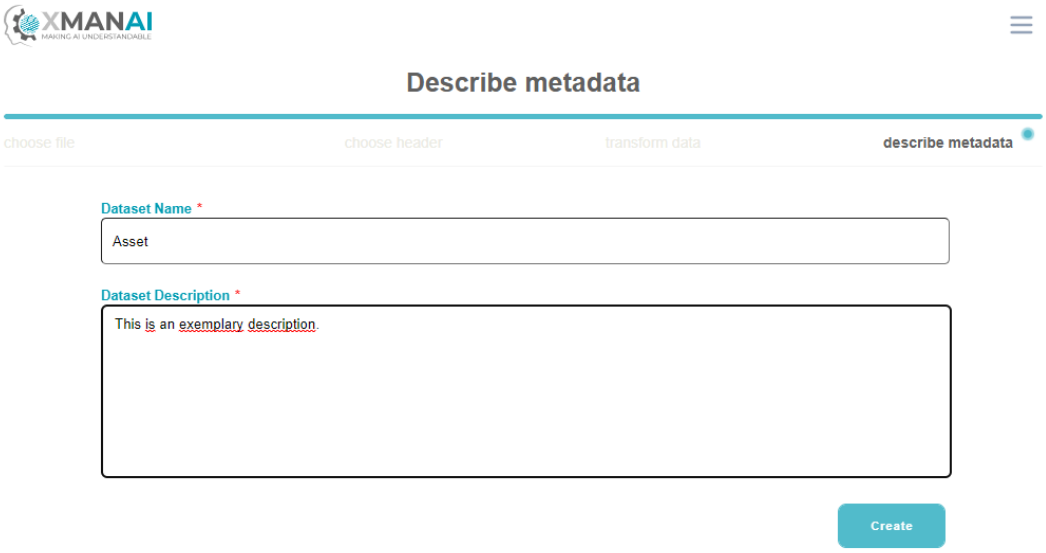


*Figure 2-7: FDH - Menu to add metadata and finally save the asset in structured format*

### 2.4.4 Technology stack and License information

The backend of the FDH is written in Python and offers its functionalities as a web service via a Flask[9] API. The frontend, which is implemented in Vue.js[10] then utilizes the backend´s endpoints. The component is published under Apache 2.0 license.

### 2.4.5 Code repository

The code of the component is published in a public section of the XMANAI repository in GitLab[11].

### 2.4.6 Improvements since the previous version

The last release didn't include the implementation of the FDH. All the above-mentioned functionalities were developed for this release.

### 2.4.7 Considerations

Currently, there are no blocking elements or special considerations to be mentioned.

## 2.5 Data Handler: File/Data Manager

### 2.5.1 Description

The File/Data manager (FDM) is the component accountable for providing an overview of all the files that a user has access to. In the XMANAI platform, there are four categories of assets: files, datasets, models, and results. FDM presents to the users an outline their files and allows basic operations, e.g. upload, rename, delete, etc. For instance, through the FDM, users can upload new files and delete or rename an existing file. Furthermore, FDM has appropriate filtering/searching features for facilitating the user in searching for a specific file. Moreover, it provides options for converting a file to a dataset, which is managed by the File Data Harvester (see Section 2.4), and exporting a file, which is done through the File Data Exporter (see Section 2.6).

For providing the above functionalities, the File/Data Manager interacts with the Identity & Authorisation Manager to identify the current user and check its credentials, with the Policy Manager to get the assets that are available to a specific user and with the Asset Store to fetch them.

The FDM supports the following functionalities:

*Table 2-7: File/Data Manager – Implemented Functionalities*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| FDM.1 | Overview of the files that are accessible to a specific user.<br>This functionality is fully supported in the current development stage of the component. |

[9] https://github.com/pallets/flask
[10] https://vuejs.org/

[11] https://GitLab.com/xmanai-h2020

| | |
|---|---|
| FDM.2 | Search in the list of the available files. The component will provide the functionality for the user to search for a specific file.<br><br>This functionality is fully supported in the current development stage of the component. The user can search and filter some of their available files based on a keyword. |
| FDM.3 | Select an available file and rename it. The user should select a file they have access to and rename it.<br><br>This functionality is fully supported in the current development stage of the component. The user can select a dataset and rename it. The FDM is responsible for interacting with the appropriate component to complete the renaming. |
| FDM.4 | Create a new file. The user will be able to upload a new file and then convert it to a dataset.<br><br>This functionality is fully supported in the current development stage of the component. There are two actions related to the creation of a new dataset. The first is about uploading a file from the user's local pc to the XMANAI application, which FDM fully supports, and the second, which is about transforming a file into a dataset, is supported by the File Data Harvester. |
| FDM.5 | Delete an available dataset<br><br>This functionality is fully supported in the current development stage of the component. The user can select a component and delete it. |

### 2.5.2 API documentation

The File/Data Manager is responsible for interacting with the user through appropriate Graphical User Interfaces (GUIs). Therefore, it does not expose any API end-point to other components.

### 2.5.3 User Interface

The figures below present the screenshots of the component's most important GUI elements.

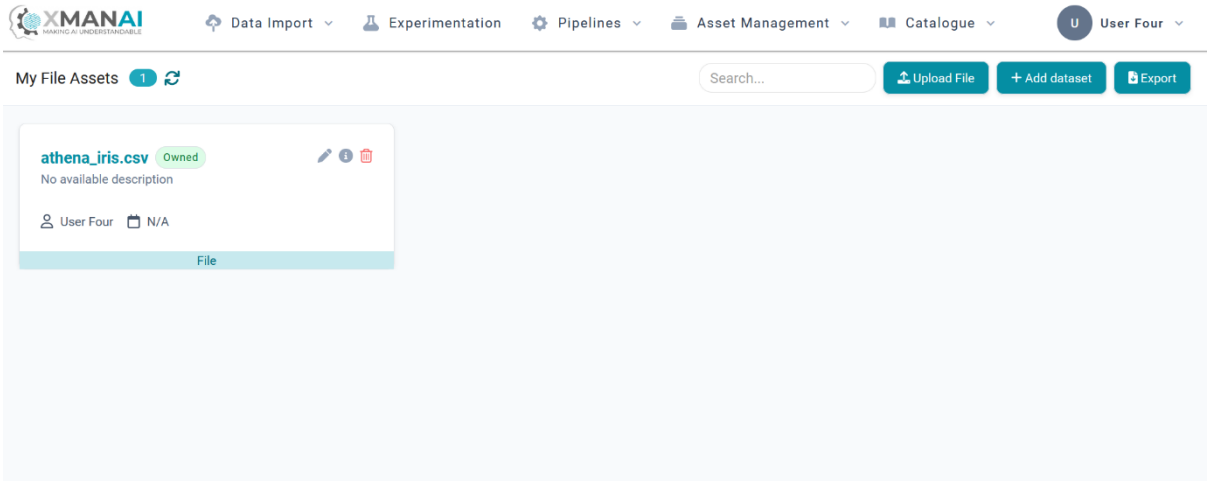


*Figure 2-8: An overview of the files of a user, as provided by the File Data Manager.*

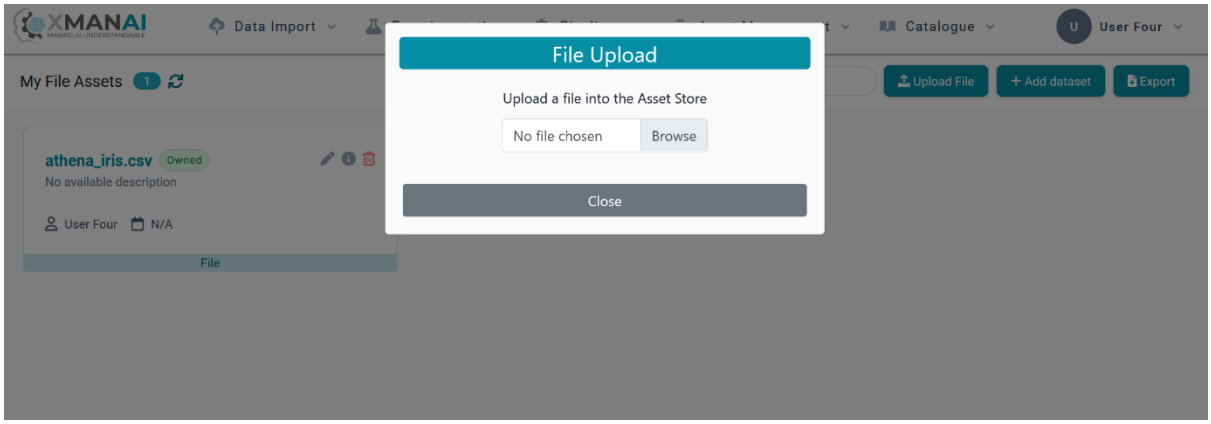


*Figure 2-9: The interface for uploading a file to XMANAI*

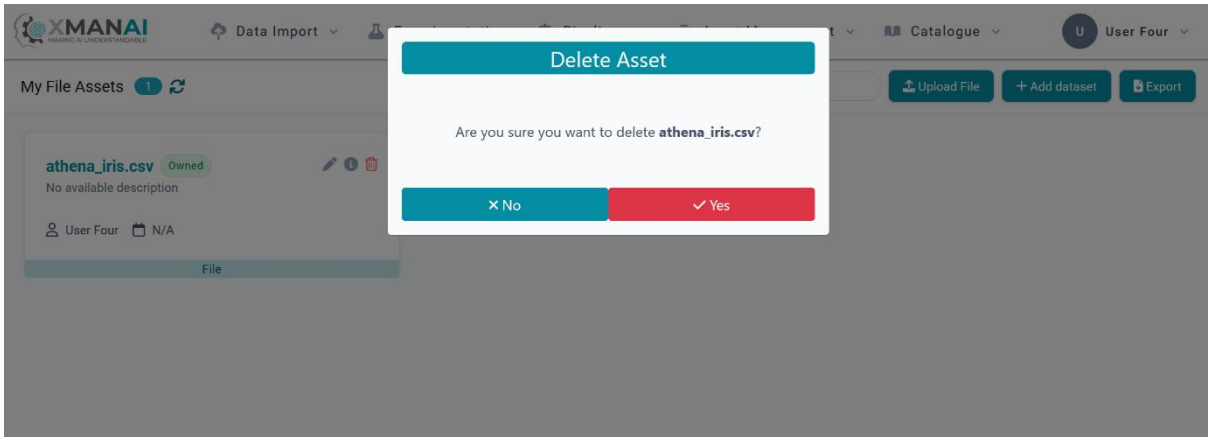


*Figure 2-10: The interface for deleting a file*

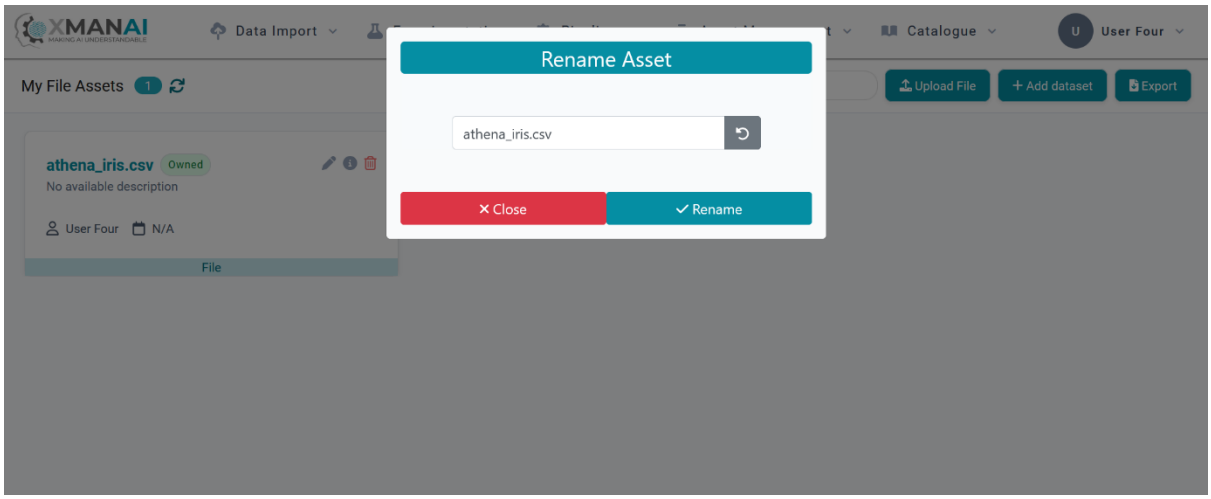


*Figure 2-11: The interface for renaming a file*

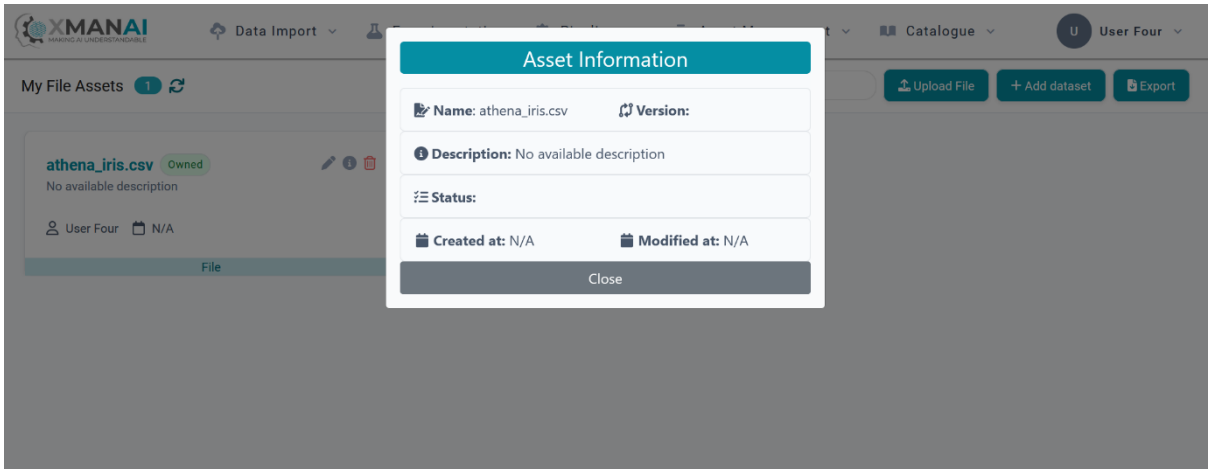


*Figure 2-12: The interface for displaying the file information. So far, there are no metadata for files and, which will be supported in the next release.*

### 2.5.4 Technology stack and License information

This section presents the technological stack for the File/Data Manager. The component is developed as a containerized web application. The back-end component is built-in on Flask (Python package) and the front-end with the Vue.js framework of the JavaScript library. The front-end and the back-end are developed inside Docker containers. This version of the component is proprietary.

### 2.5.5 Code repository

The code of the component is published in a private section of the XMANAI repository in GitLab[12], access to which can be provided by a request sent to the project coordinator.

### 2.5.6 Improvements since the previous version

The most significant improvements since the previous version are related to the integration with the XMANAI platform. In the previous version, FDM was not fully integrated, and many functionalities were limited to standalone mode. To address this, we have completed the integration points with the appropriate components: with the (a) identity and authorization manager for the login mechanism, (b) the policy manager for fetching the list of the user's files, and (c) the asset store for file retrieval.

### 2.5.7 Considerations

The FDM seems to fully support the appropriate functionalities, so there are no major consideration points.

## 2.6 Data Handler: Data Exporter

### 2.6.1 Description

[12] https://GitLab.com/xmanai-h2020

The Data Exporter is responsible for all the data local saving functionalities. It mainly creates slices of the data that are gathered into the XMANAI platform and saves file instances of them in different accessible formats (e.g.., textual, tabular or binary). The data exporter will cover different XMANAI layers of internal data sources, which include datasets.

As it is mentioned in D2.1, the Data Exporter is expected to be used by all XMANAI user roles: business users, mainly to retrieve predictions and explanations, as well as data scientists and engineers, mainly to manage slices of the datasets. The table below lists the component functionalities provided in the current release.

*Table 2-8: Data Exporter – Implemented Functionalities*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| DE.1 | Select whether to access the dataset of the project or a set of prediction results (along with their explanations) previously generated. For this deliverable, as we do not have predictions at this moment, only the access to export datasets is provided. Concretely this functionality provides:<br>• Access to different datasets.<br>• The list of available datasets as a table of available datasets.<br>• Capability to select a dataset from the list of available datasets |
| DE.3 | **Convert the queried data into a local file.** This functionality allows the user to select an output format for the data that it is intended to export and save the generated file into a specified path. Concretely, this functionality provides:<br>• The ability to select between different output formats. For this release, as the initial dataset will be in tabular format, CSV and XLSX formats are provided.<br>• The ability to slice the datasets by row or by columns, selecting only what the user really wants.<br>• The capability to select a path to store the file into the local system.<br>• The generation of a file with the data of the dataset using the specified output format. |

### 2.6.2 API documentation

The documentation of the component's API is available online[13] and summarised in the table below:

*Table 2-9: Data Exporter – API Endpoints Overview*

| Endpoint | Method | Description |
|---|---|---|
| /api/files | GET | Retrieve all files that can be exported from this component |
| /api/files/formats | GET | It returns the list of available formats for the formatting of datasets |

[13] https://data-exporter-backend.ai4manufacturing.eu/api/

| | | |
|---|---|---|
| /api/files/{id} | GET | Get the exported dataset, formatted and sliced based on the given query arguments<br>• From (optional): the initial format of the file<br>• to: the new format of the file. Must be one of the formats returned by the previous endpoint<br>• columns (optional): The comma separated list of columns that must be in the exported version of the dataset<br>• starting_row (optional): The starting row for slicing the dataset row-wise.<br>• end_row (optional): The last row for slicing the dataset row-wise. |

### 2.6.3 User Interface

The screenshot of the component's GUI is presented on the figure below.

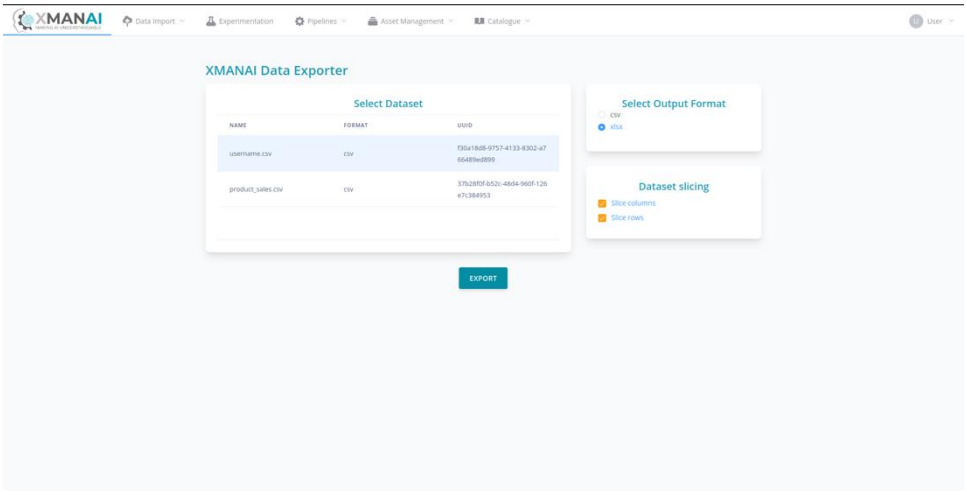


*Figure 2-13: The main Data Exporter Interface. The user chooses the dataset that they want to export.*

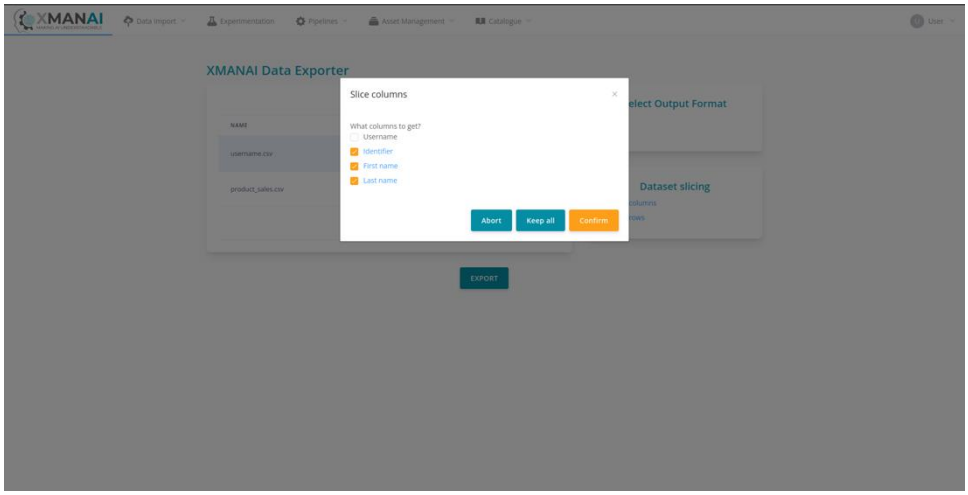


*Figure 2-14: The interface for slicing the columns of the tabular dataset*

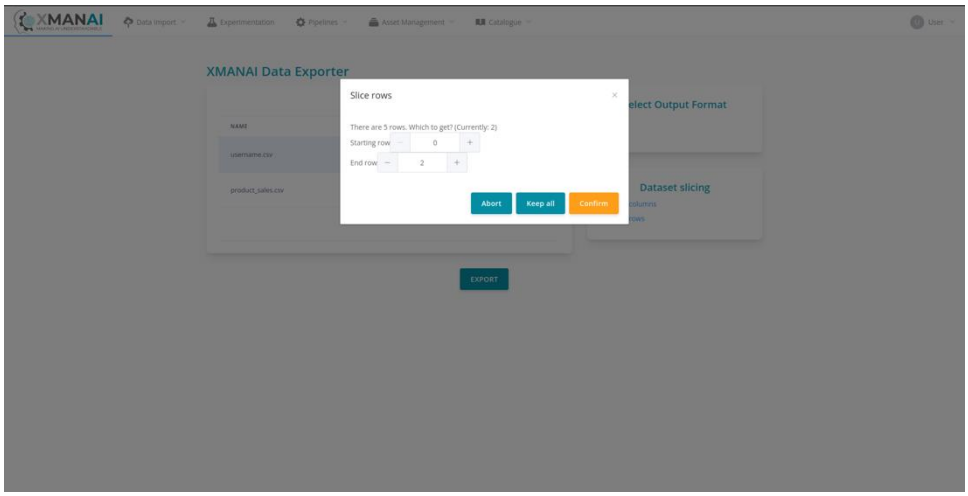


*Figure 2-15: The interface for slicing the rows of the dataset, with the range of first-last row index*

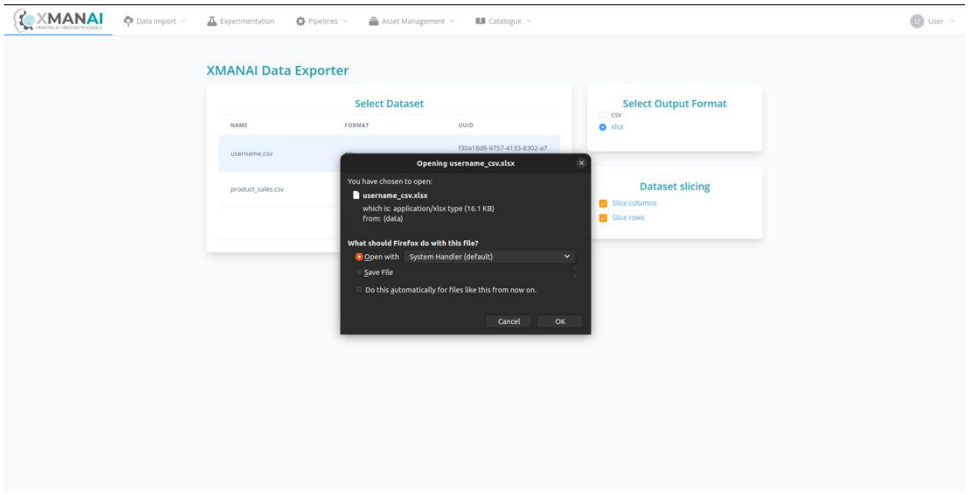


*Figure 2-16: The exported file to be downloaded by the user. Although the original format of the dataset was csv, we can see that the final format has been converted to .xlsx*

### 2.6.4 Technology stack and License information

This component executes different Docker containers in order to deploy it. Concretely, two containers have been deployed:

- Front-end. This container is employed to visualize the user interface. For this part, Nuxt.js has been used.
- Back-end. This container uses the backend for its slicing and format converting capabilities. We found Node.js to be a good tool for this.
- Nginx. It serves as a proxy and directs client requests to the appropriate component.

This component has a closed source license and an agreement with Tyris must be reached in order to use it. Nuxt.js and Node.js are Open Source with an MIT license. Nginx is Open Source too, but it has a FreeBSD license.

### 2.6.5 Code repository

The code of the component is published in a private section of the XMANAI repository in GitLab[14], access to which can be provided by a request sent to the project coordinator.

### 2.6.6 Improvements since the previous version

Since the previous release, there have been major improvements in usability in the Data Exporter Component. The data exporter now allows for the user to select the output format of the converted files from the available ones, rather than downloading the CSV stored in the assets store. We now allow the user to slice the data by rows and by columns, allowing them to select the exact columns that they want, and a range of the dataset rows, defined by a start – end index range. This is especially important for big datasets, where sampling a smaller version of the file can provide insights about its contents, and there is no need to fetch the entire file.

### 2.6.7 Considerations

As stated in the functionalities and implementation status, only the access to export datasets is provided.

## 2.7 XAI Marketplace: Registry/Metadata Manager

### 2.7.1 Description

The scope of the Registry/Metadata Manager is to store and maintain the metadata information that is used to extensively describe every AI artefact (i.e. dataset, AI model, result), existing in the XMANAI platform, as well as to provide the appropriate management interface for the user to clearly define or update this information. Through the same interface, the user is allowed to share the AI artefact with other departments within or beyond their organization inside the XAI Marketplace. At a higher level, the Registry/Metadata Manager functions as a catalogue where a data consumer can browse the available assets, search for specific asset-related information or filter out items of little interest.

All metadata information is stored in the Metadata Store, which communicates directly with the Metadata Manager, and adheres to the XMANAI metadata schema with the following features:

[14] https://GitLab.com/xmanai-h2020

*Table 2-10: XAI Marketplace Metadata Schema*

| Metadata Name | Definition | Category | Related Standard(s) |
|---|---|---|---|
| **Identifier** | A unique identifier of the asset. | Dataset, Model, Result, File | DCMI, DCAT |
| **Title** | A name given to the specific asset. | Dataset, Model, Result, File | DCMI, DCAT |
| **Description** | A free-text account of the specific asset. | Dataset, Model, Result | DCMI, DCAT |
| **Type** | The nature or genre of the asset (i.e. dataset, model, ...) | Dataset, Model, Result, File | |
| **Provider** | The entity responsible for making the asset available or to whom the asset belongs. | Dataset, Model, Result, File | DCMI, DCAT |
| **Keywords** | A keyword or tag describing the asset. | Dataset, Result | - |
| **Temporal Coverage** | The temporal period that the dataset covers. | Dataset, Result | DCMI, DCAT |
| **Spatial Coverage** | The geographical area covered by the dataset. | Dataset, Result | DCMI, DCAT |
| **Purpose** | The problem which the AI solution was created to address. | Model, Result | - |
| **Algorithm** | The ML/DL algorithm that was used for this model. | Model | - |
| **Retrainable** | An indication whether the specific model can be trained again. | Model | - |
| **Task** | The main task (e.g. classification, clustering) that the model is targeting to solve. | Model | - |
| **Date Created** | The date when the asset was created/released. | Dataset, Model, Result, File | DCMI, DCAT |
| **Date Updated** | Most recent date on which the asset was changed, updated or modified. | Dataset, Model, Result | DCMI, DCAT |
| **Access Rights** | An indication of the asset's security status. | Dataset, Model, Result | - |
| **License** | A license, a legal document under which the asset is made available. | Dataset, Model, Result | DCMI, DCAT |
| **Version** | The version of the specific asset as well as any other provenance info retrieved from the Provenance Engine. | Dataset, Model, Result | DCMI |
| **Structure** | The columns of the specific asset (including their title and data type) | Dataset, Result | - |

Most of these metadata details are directly provided by the user, while a few of them can be computed automatically (e.g., current version, date updated, algorithm, retrainable, task, etc.) or be provided in collaboration with other XMANAI components, such as the Data Harvester, the XAI Pipeline Manager, the XAI Models Catalogue or the Provenance Engine. It should be noted that the *access rights* feature defines whether the asset can be shared or not in the marketplace in collaboration with the Data Access Manager.

The following table summarizes the implemented features in the final release.

*Table 2-11: Registry/Metadata Manager – Implemented Functionalities*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| RMM.1 | **Data asset metadata management.** This functionality**,** currently, involves metadata stored in the Metadata Store for datasets, AI models, results and files and can be further analysed into four management subfunctions:<br>• Initial definition/creation of metadata for an asset<br>• Enrichment or update of metadata for an asset<br>• Retrieval of metadata for an asset<br>• Removal of metadata for an asset |
| RMM.2 | **Data asset catalogue browsing.** This functionality provides an interface for the potential data asset consumers to investigate and examine in detail the various AI-related assets, as part of a catalogue. The data assets presented to the user have been automatically filtered based on the respective access policies and IPRs, which are communicated by the Policy Engine. The eligible assets can be further examined separately, exposing information that depends again on the access level of the viewer. For example, if the data asset belongs to the same organisation as the user, the view will be much more informative, whereas an external (to the department/organisation) user will view only basic details. |
| RMM.3 | **Data asset search and discovery.** This functionality allows a data asset consumer to locate assets of interest or search for very specific properties using:<br>• Text search<br>• Filtering<br>• Sorting |

## 2.7.2 API documentation

The Registry/Metadata Manager provides a complete toolset of explicit APIs that allow the consumption of the provided functionalities by the rest of the XMANAI components. These endpoints are documented and can be found via the Swagger documentation library in the public part of the project's code repository[15]:

*Table 2-12: Registry/Metadata Manager – API Endpoints Overview*

| Endpoint | Method | Description |
|---|---|---|
| /api/v1/marketplace/assets | POST | Create a new data asset (dataset, model, result, file) and store its accompanying metadata. |
| /api/v1/marketplace/data-assets/uuid/{uuid} | GET | Retrieve the metadata of a data asset (dataset, file, model, file) by its uuid. |
| /api/v1/marketplace/data-assets/pipe/{pipeid} | GET | Retrieve the metadata of a data asset (dataset, file, model, file) by the id of the pipeline that created it. |
| /api/v1/marketplace/assets/uuid/{uuid} | PUT | Update the metadata of a specific data asset. |

[15] https://GitLab.com/xmanai-h2020/api-docs/-/blob/main/XAI-Marketplace-1.0.0-resolved.yaml

| Endpoint | Method | Description |
|---|---|---|
| /api/v1/marketplace/assets/uuid/{uuid} | DELETE | Delete the metadata of a data asset by its id. |

### 2.7.3 User Interface

As depicted in the following figure, the user visits the XAI Marketplace where he/she may view the available assets depending on their category along with their basic information (title, description, provider, type). He/she may also search for datasets through the free text search functionality while sorting or filtering the results based on different parameters.

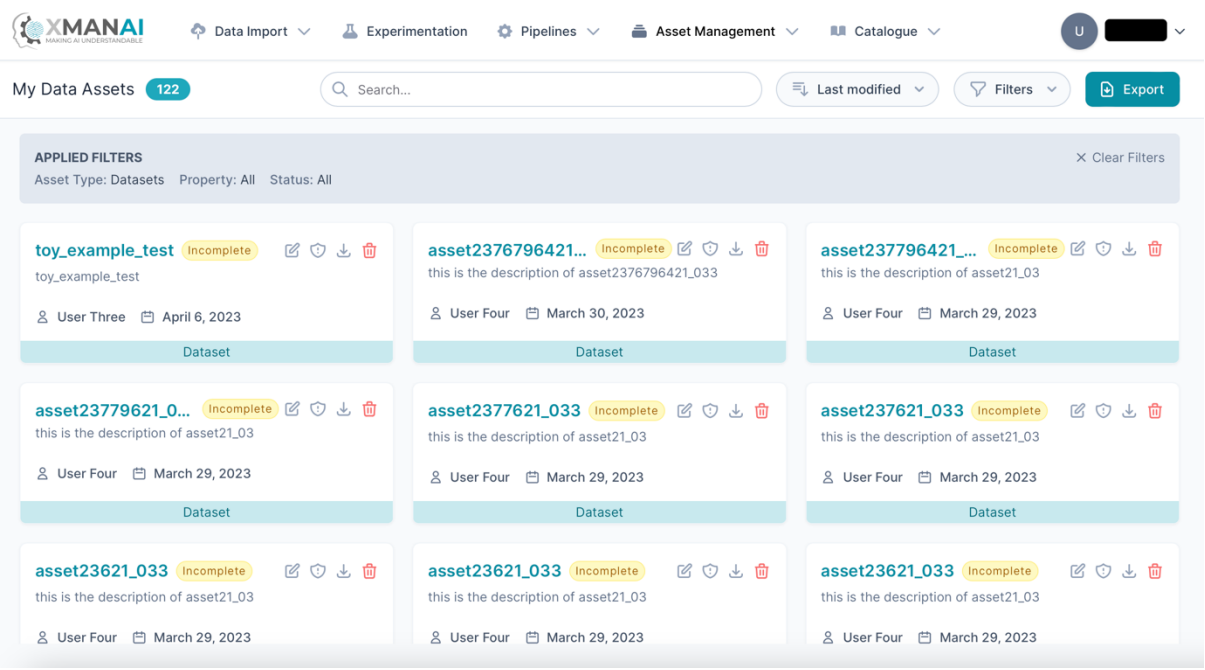


*Figure 2-17: XAI Marketplace - Registry/Metadata Manager – My Data Assets. Main Page*

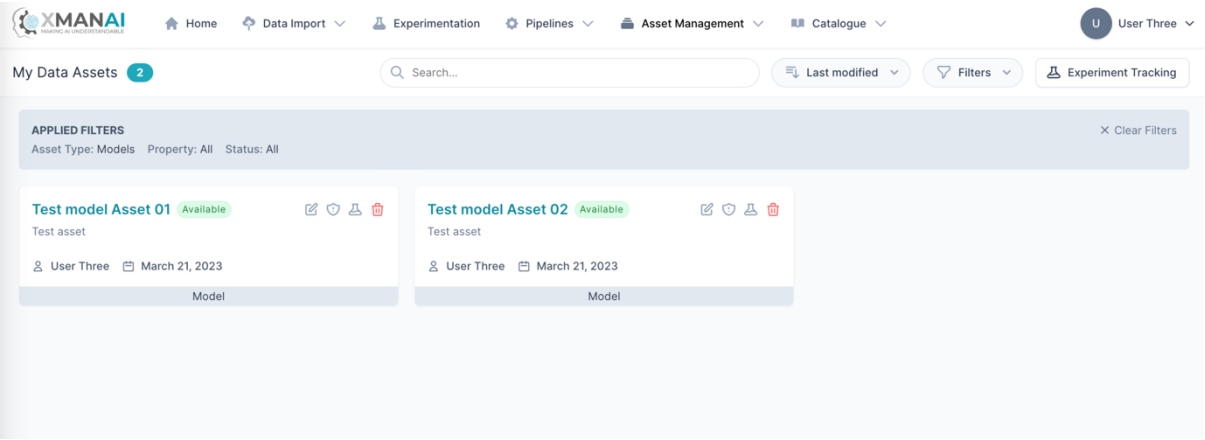


*Figure 2-18: XAI Marketplace - Registry/Metadata Manager – My Models. Main Page*

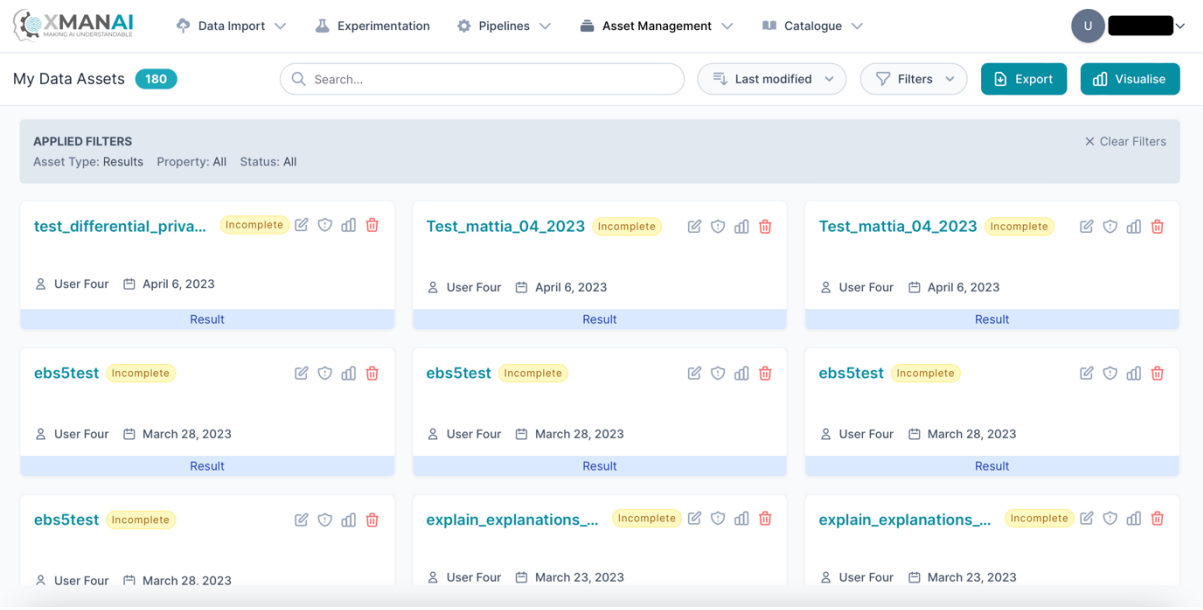


*Figure 2-19: XAI Marketplace - Registry/Metadata Manager – My Results. Main Page*

By selecting a specific asset, the user may view its detailed metadata depending on its type, as depicted in the following figures.

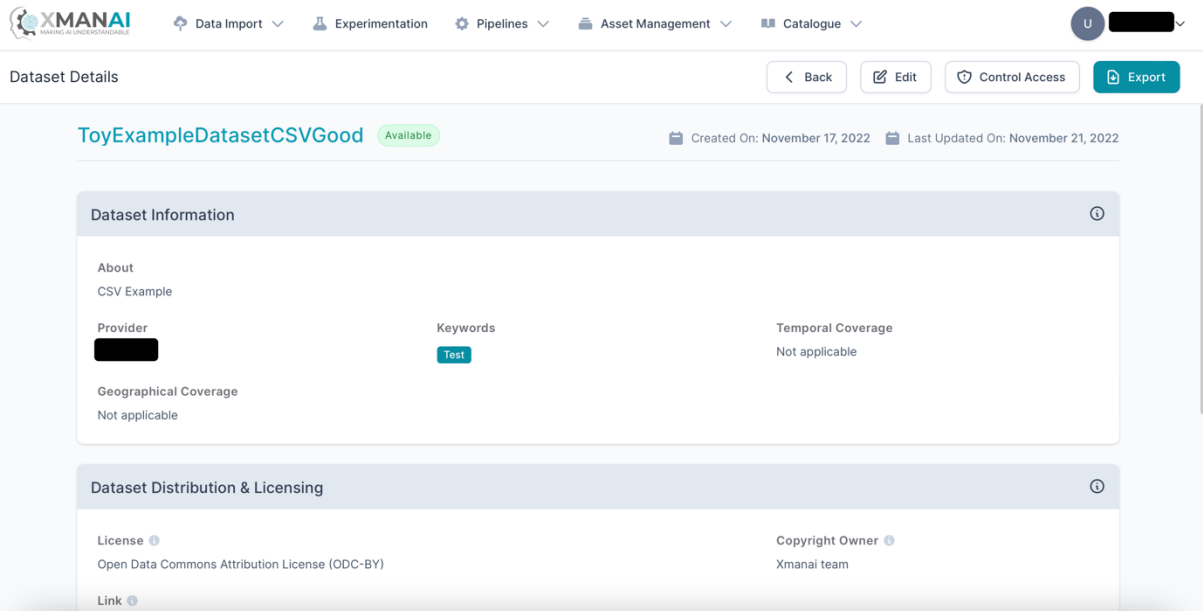


*Figure 2-20: XAI Marketplace - Registry/Metadata Manager. View Dataset Details*

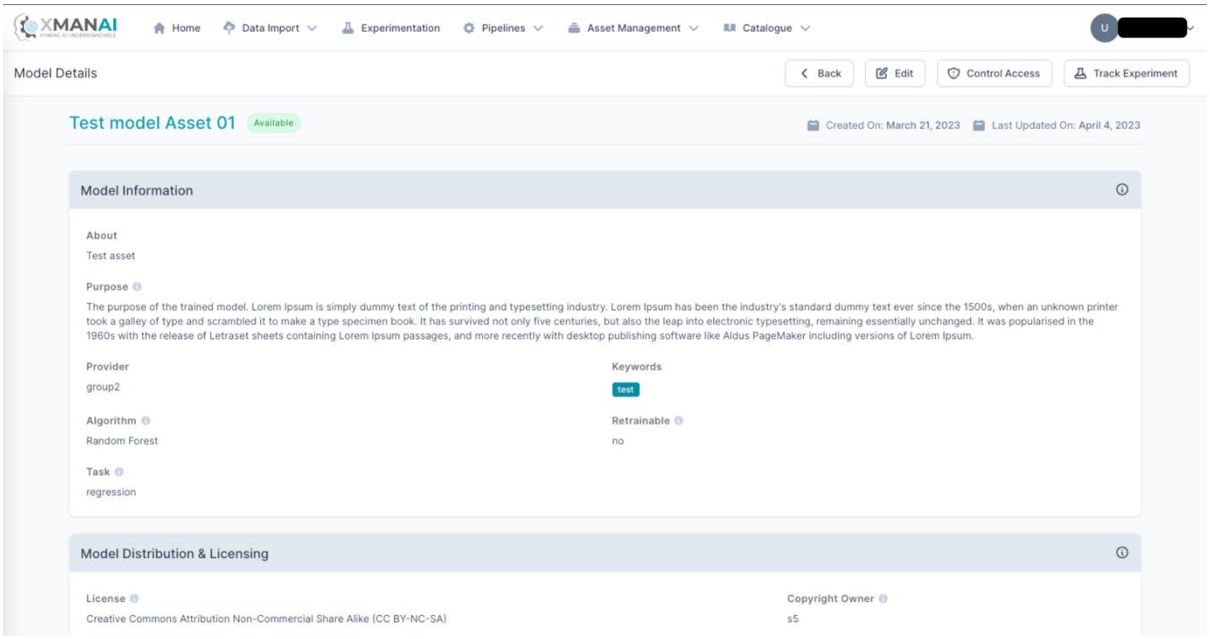


*Figure 2-21: XAI Marketplace - Registry/Metadata Manager. View Model Details*

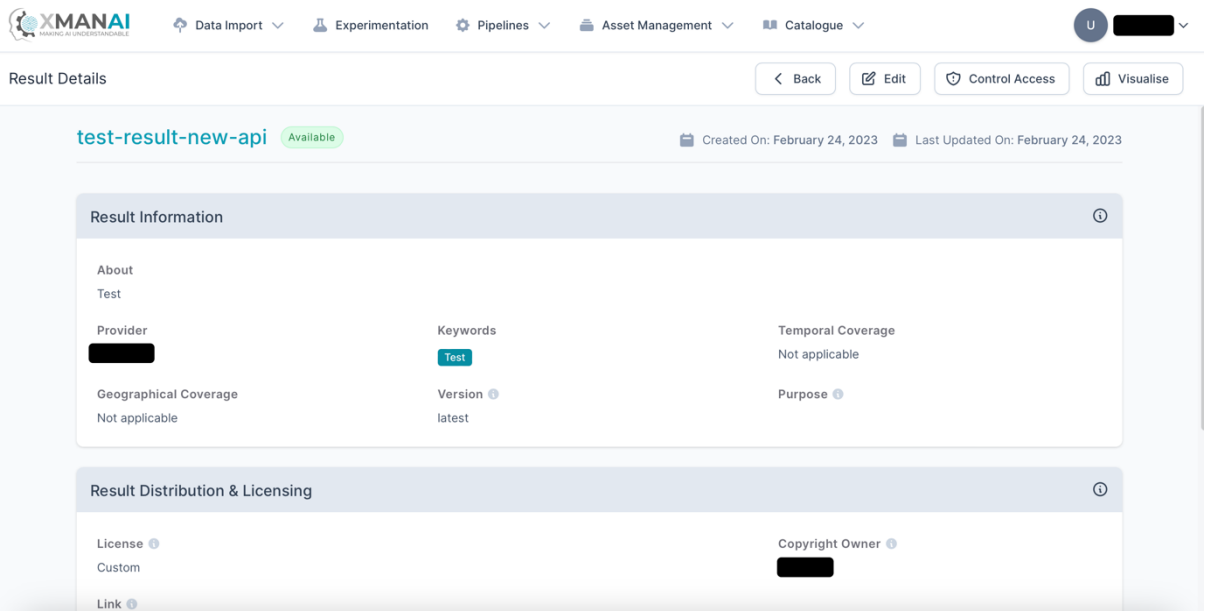


*Figure 2-22: XAI Marketplace - Registry/Metadata Manager. View Result Details*

As users are able to share their assets in order to collaborate with other stakeholders from external organizations (to their own) or acquire missing data, they come across the different assets that have been shared at cross-organization level and they (as users under a specific organization) are eligible to view (according to the assets' access policies) as depicted in the following figure.

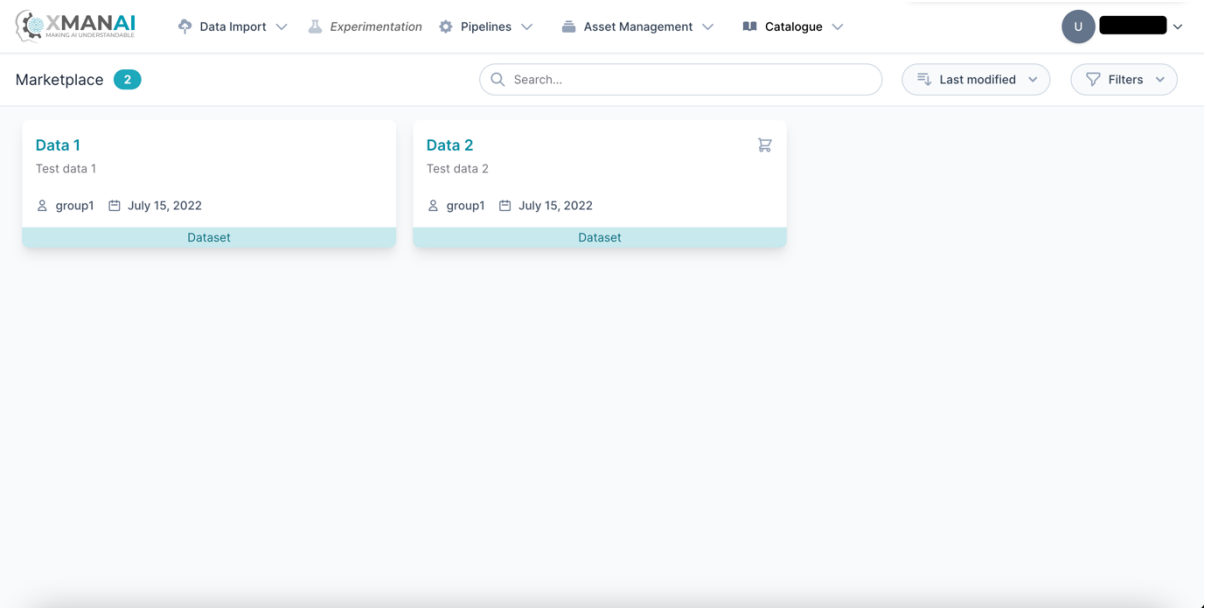


*Figure 2-23: XAI Marketplace – Search for assets*

### 2.7.4 Technology stack and License information

The backend implementation of the Registry/Metadata Manager component is solely based on the S5 Share Platform, which is a proprietary, closed source product. In XMANAI, the S5 Share Platform has been updated to support the adopted metadata schema (as presented in Table 2-10: XAI Marketplace Metadata Schema) and is extended to support additional data assets. The user interface (UI) has also been redesigned and is developed using the Vue.js[16] JavaScript framework, released under the MIT License.

### 2.7.5 Code repository

The code of the component is published in a private section of the XMANAI repository in GitLab, access to which can be provided by a request sent to the project coordinator.

### 2.7.6 Improvements since the previous version

In the final release of the Registry/Metadata Manager, a set of improvements have been introduced (in addition to bug fixes and UI improvements):

- Managing the metadata of two additional asset types is now supported: (a) Results that represent the outcome of XAI pipelines, as already anticipated in the XMANAI Deliverable D2.2; (b) Files that are handled through the File/Data Manager and were identified as an additional need (in respect to D2.2) in the XMANAI Platform.
- Additional metadata have been identified and included in the XAI Marketplace Metadata Schema for the new asset types, but also for the existing asset types (e.g. the Structure field for datasets, the Algorithm or Retrainable fields for models).

[16] https://vuejs.org/

### 2.7.7 Considerations

At the time D2.2 was written, XAI pipelines and experiments were considered as AI assets that were within the scope of the Registry/Metadata Manager. However, in the course of integration activities, it was acknowledged that XAI pipelines, explanations and experiments are not to be shared on their own (as independent assets) across organizations. In addition, the relevant information that accompanies XAI pipelines, explanations and experiments is automatically inferred and thus should be directly handled through the relevant components of the XMANAI Platform (i.e. XAI Pipeline Designer, XAI Model Explanation Engine, Experiment Tracking Engine).

Taking into consideration that the integration activities are ongoing for the beta release of the XMANAI Platform on M32 and that the final version of all Services Bundles has been released at the same time that the alpha phase of the demonstration activities (D6.4) concluded, additional updates (e.g. additional metadata for all asset types) may occur (during the WP5 activities) in the Registry/Metadata Manager to address any additional needs that may emerge.

## 2.8 XAI Marketplace: Contract Manager

### 2.8.1 Description

The role of the Contract Manager is to facilitate asset sharing between different organizations, their departments, and their users. It is the core XAI Marketplace component, which handles all operations regarding smart contracts, contract terms enforcement and the interaction between end users concerning agreement signing, negotiations and asset transfer.

In this context, the Contract Manager provides the required functionalities and interface for the creation and modification of a contract, as well as for monitoring its current status since several factors can affect it. For example, a contract may be set to expire after a set period of time. Similarly, a contract may have a number of negotiation rounds and end up being cancelled if the involved parties reach no agreement. All of these activities occur in a semi-automatic manner, so that the final decision making is performed by the involved humans.

The functionalities supported by the Contract Manager can be summarized as follows:

*Table 2-13: Contract Manager – Implemented Functionalities*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| SCM.1 | **Data asset sharing through smart contracts.** When an acquisition request is made from a data asset consumer, a draft smart contract is automatically produced on behalf of the data asset provider using as a basis the information stored in the asset's metadata. Of course, the data asset provider is able to modify it prior to sending it to the data consumer, if so wishes. Then, a phase of negotiation commences that can be resolved outright or take some time if several changes on the proposed terms are progressively requested by either party. The process continues until all involved parties agree on all terms or either party decides to cancel the contract. In the case of a positive outcome, the involved parties digitally sign the contract, which results in its activation (taking into consideration that payments are not foreseen in XMANAI). In each step of the process, the Contract Manager communicates with the XMANAI distributed ledger to securely store the smart contract details. |
| SCM.2 | **Smart contract enforcement.** This functionality ensures that no violation of the contract terms occur at any given moment from the involved parties in the XMANAI Platform. To achieve this, the Contract Manager assesses all actions on the asset(s) involved and protects against any processing or manipulation against the agreed terms. The Contract Manager also communicates with the blockchain in order to verify whether a specific contract is: a) valid and b) active. This is a necessary step when a user attempts to use an acquired dataset, since it must be ensured |

| Feature ID | Functionalities & Implementation Status |
|---|---|
| | that an active asset contract is in place and at the same time, access is still granted and has not expired. |
| SCM.3 | **Smart contract export.** All involved parties should be able to retrieve (i.e. download) the agreed terms and conditions as a file for archiving purposes but also to better study the contract's content and receive approval from their legal department. |

### 2.8.2 API documentation

The API endpoints for the Contract Manager component are documented and can be found via the Swagger documentation library in the project's code repository[17].

*Table 2-14: Contract Manager – API Endpoints Overview*

| Endpoint | Method | Description |
|---|---|---|
| /api/v1/marketplace/contracts | GET | Retrieve the current organisation's contracts. |
| /api/v1/marketplace/contracts | POST | Create a new contract. |
| /api/v1/marketplace/contracts/active-pending | GET | Get a list of contracts that are active or in progress. |
| /api/v1/marketplace/contracts /{id} | GET | Retrieve a specific contract by its id. |
| /api/v1/marketplace/contracts/active /{id} | GET | Check if a specific asset has an active contract. |
| /api/v1/marketplace/contracts/{id} /negotiate | PUT | Negotiate a specific contract. |
| /api/v1/marketplace/contracts/{id} /sign | PUT | Sign a specific contract. |
| /api/v1/marketplace/contracts/{id} /accept | PATCH | Accept the terms of a specific version of a contract. |
| /api/v1/marketplace/contracts/{id} /reject | PATCH | Reject the terms of a specific version of a contract. |
| /api/v1/marketplace/contracts/{id}/pdf | GET | Export a contract as a pdf file. |
| api/v1/marketplace/assets/check-access | POST | Check if a user has access to specific assets. |
| api/v1/marketplace/assets/types/{{ typeID}}/active-contracts | GET | Provide a list of all assets of a specific type (datasets, models, results) for which a user has an active contract. |

### 2.8.3 User Interface

[17] https://GitLab.com/xmanai-h2020/api-docs/-/blob/main/XAI-Marketplace-1.0.0-resolved.yaml

The user may navigate to the list of contracts to which his/her organization is involved while being able to search for specific assets that have a contract in any phase, soft based on the latest contracts or based on title (ascending/descending) and filter a specific contract phase.

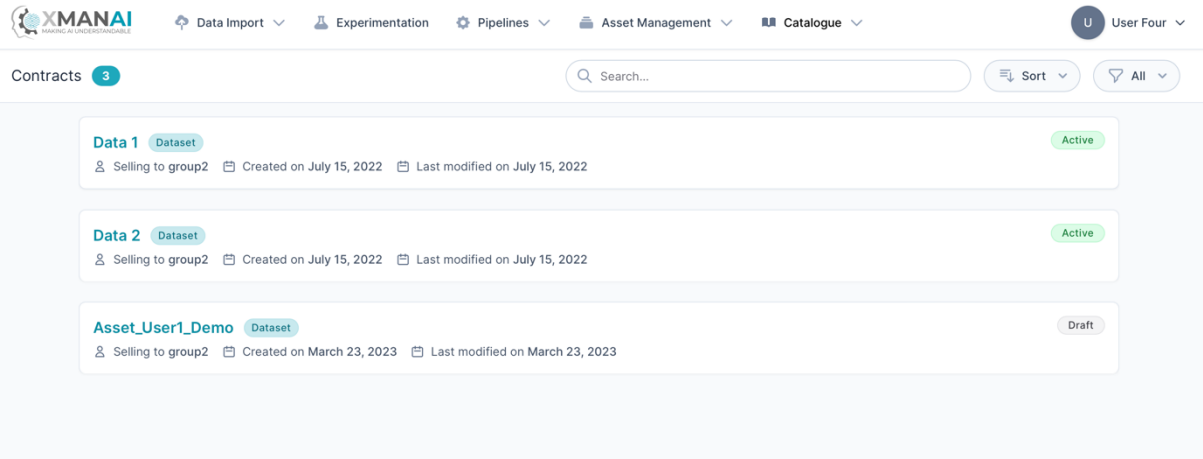


*Figure 2-24: XAI Marketplace - Contract Manager. View Contracts List*

If a user finds an asset in the Marketplace (Figure 2-23), he/she is interested in using or collaborating over, he/she initiates a request and provides a relevant message to the asset owner as depicted in Figure 2-25.

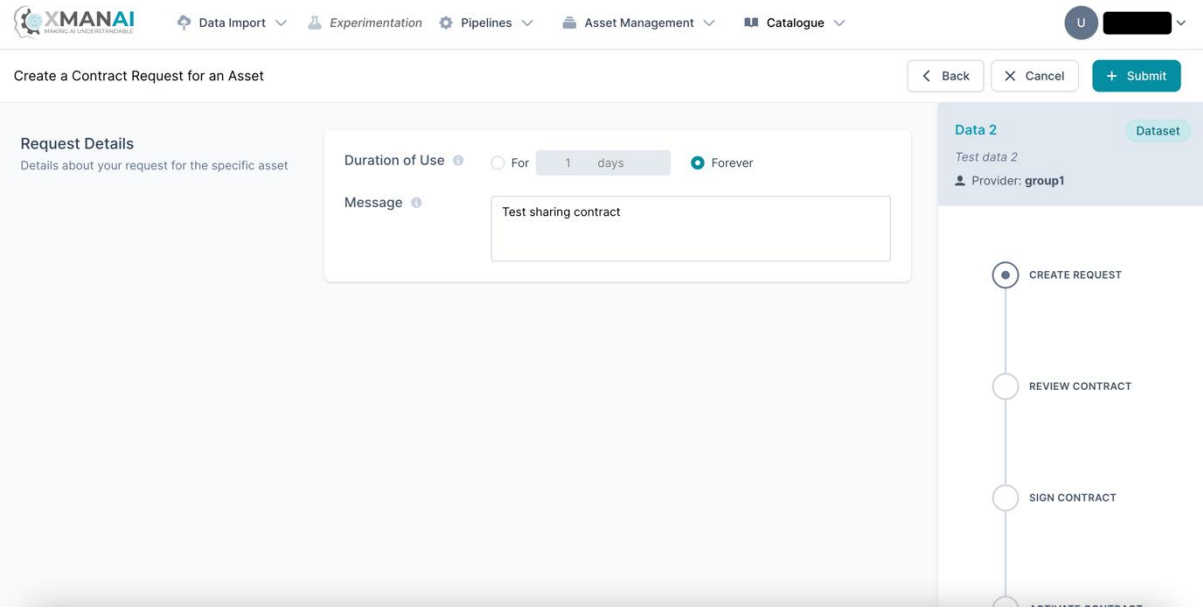


*Figure 2-25: XAI Marketplace - Contract Manager. Request for an asset*

The asset owner may review such a request and take appropriate action by offering a sharing agreement or declining the request as depicted in Figure 2-26.

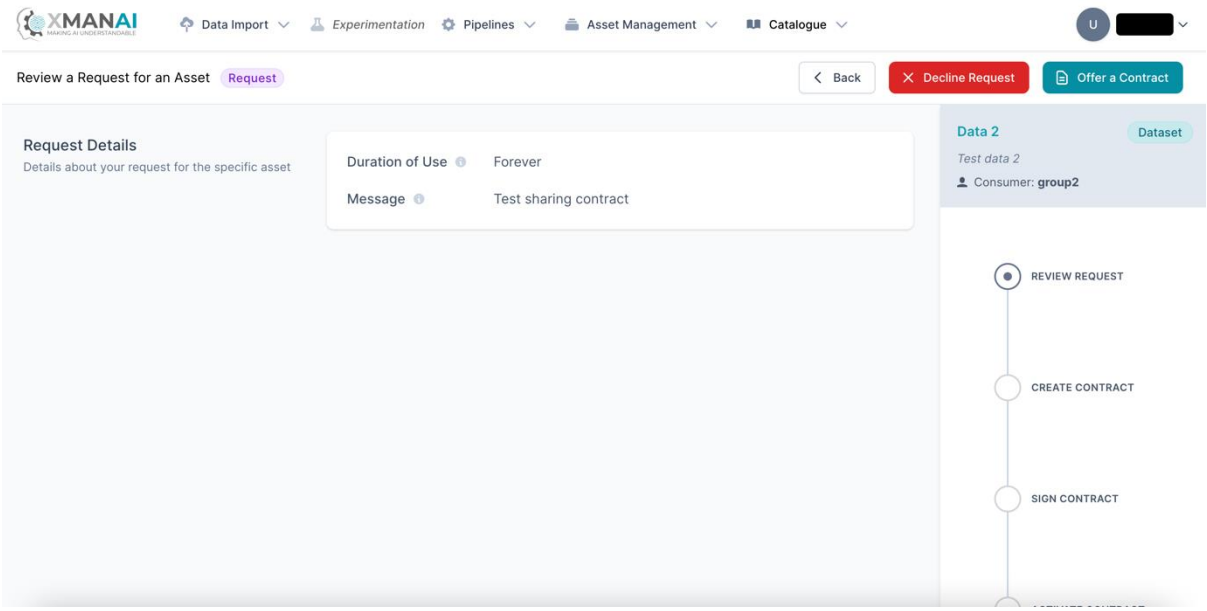


*Figure 2-26: XAI Marketplace - Contract Manager. Review the request for an asset*

If the asset owner opts to proceed with sharing an asset, he/she prepares a draft sharing agreement in which he/she defines the terms of use and defines whether the interested user will have read or write access to the asset (Figure 2-27).

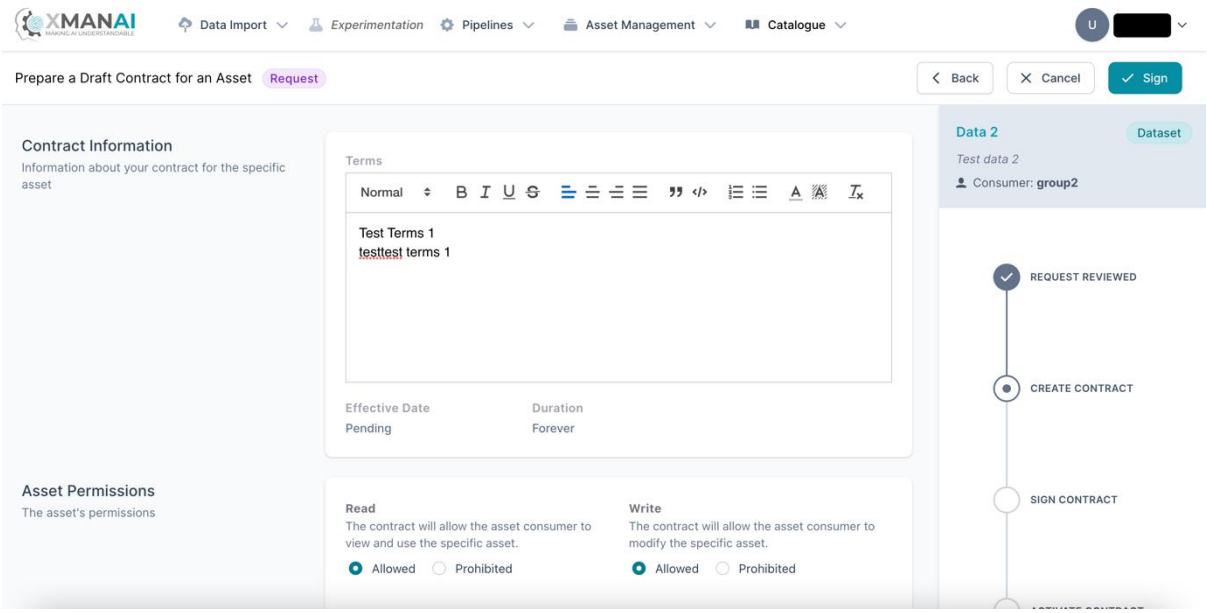


*Figure 2-27: XAI Marketplace - Contract Manager. Prepare a draft sharing agreement for an asset*

The asset owner (note: only the manager of the organization, as described in the XMANAI Deliverable D5.2) needs to sign the draft sharing agreement, so he/she needs to provide the credentials to his/her organization's DLT (Distributed Ledger Technologies / Ethereum) wallet. Once the wallet is unlocked, the sharing agreement is mined and considered as signed.

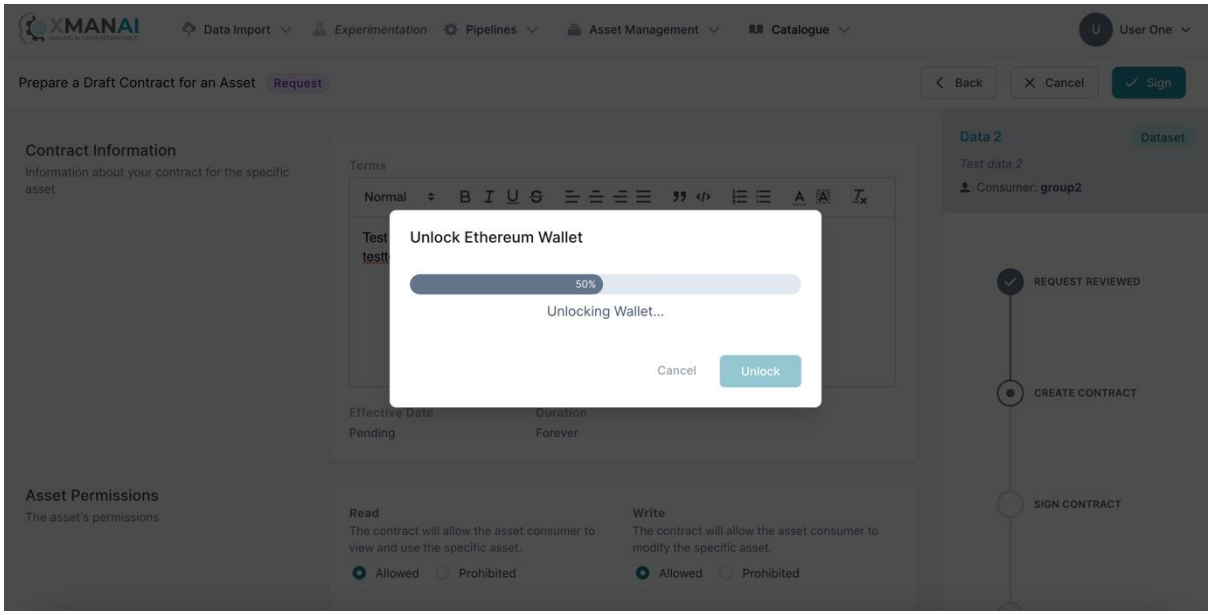


*Figure 2-28: XAI Marketplace - Contract Manager. Sign the draft sharing agreement for an asset*

The interested party may then review the draft sharing agreement and decide how to proceed: (a) accept the agreement, (b) negotiate over the terms, (c) reject the agreement (Figure 2-29).

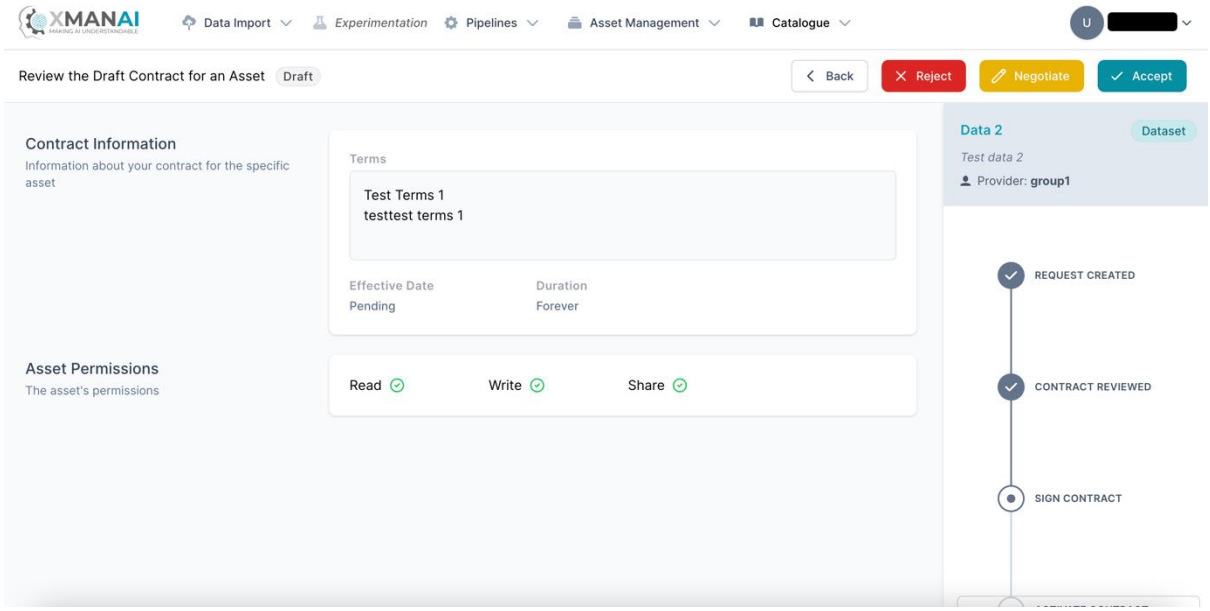


*Figure 2-29: XAI Marketplace - Contract Manager. Review the draft sharing agreement for an asset*

If the interested party opts for negotiation, he/she may amend the terms of use and/or revise the permissions allowed over the asset as depicted in Figure 2-30. The sharing agreement is again signed, by the interested party this time, and the transaction is mined in the XMANAI Blockchain.

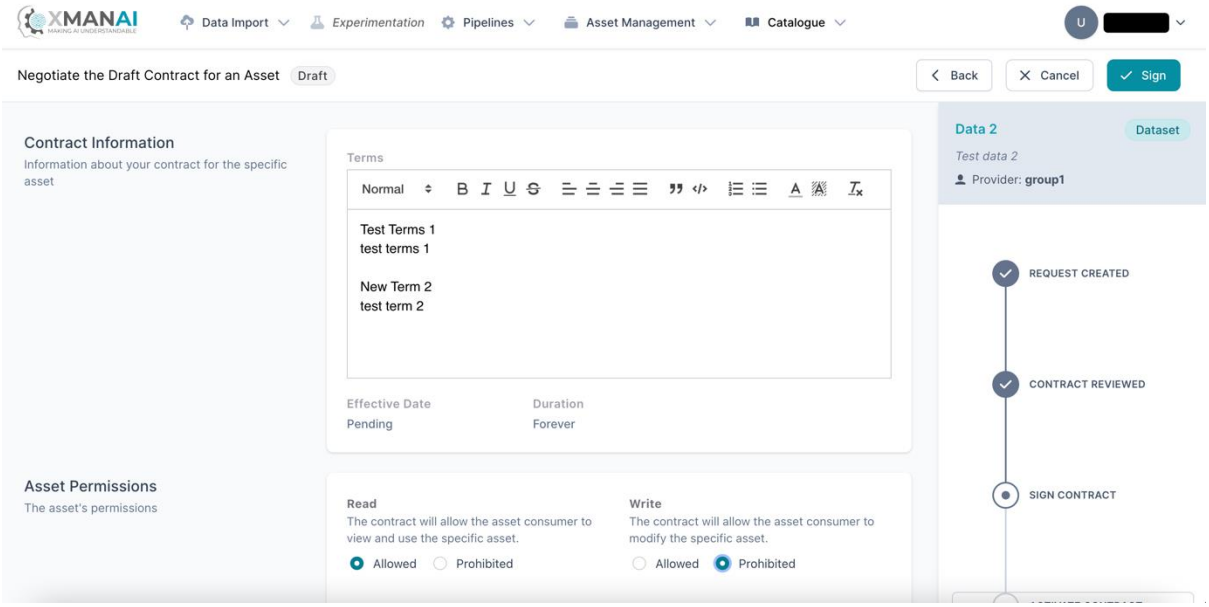


*Figure 2-30: XAI Marketplace - Contract Manager. Negotiate a draft sharing agreement for an asset*

The asset owner is then able to review the revised sharing agreement for an asset while obtaining a quick overview of the changes introduced (Figure 2-31).

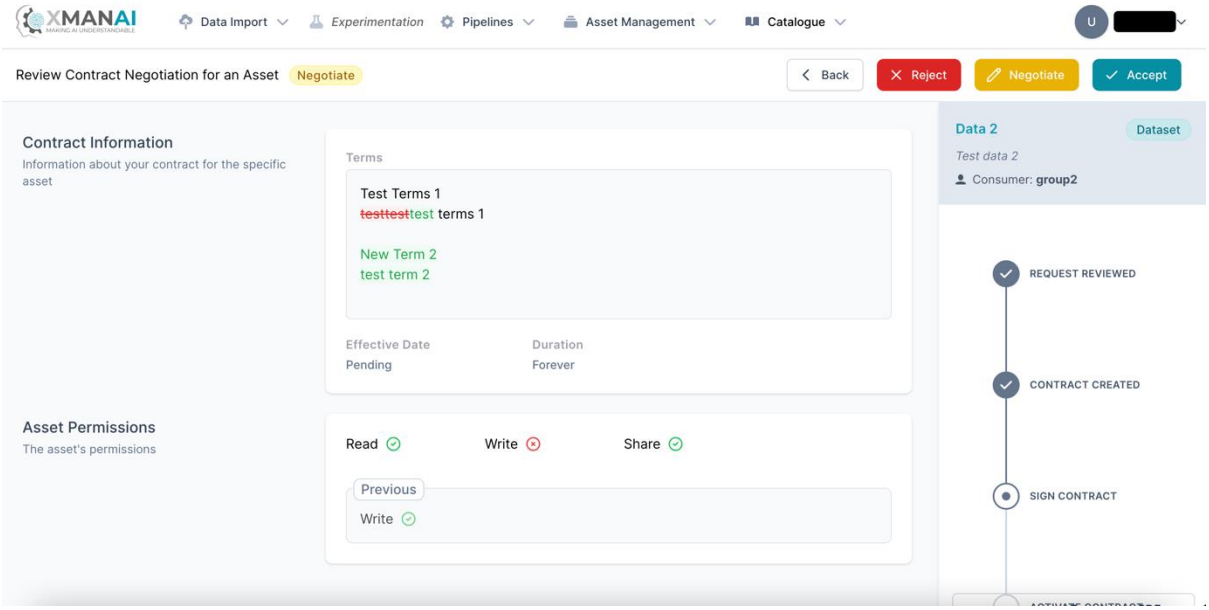


*Figure 2-31: Catalogue – Review a revised sharing agreement for an asset*

Once all involved stakeholders agree on the sharing contract, it is considered as active and the interested party obtains appropriate access to the asset (Figure 2-32).

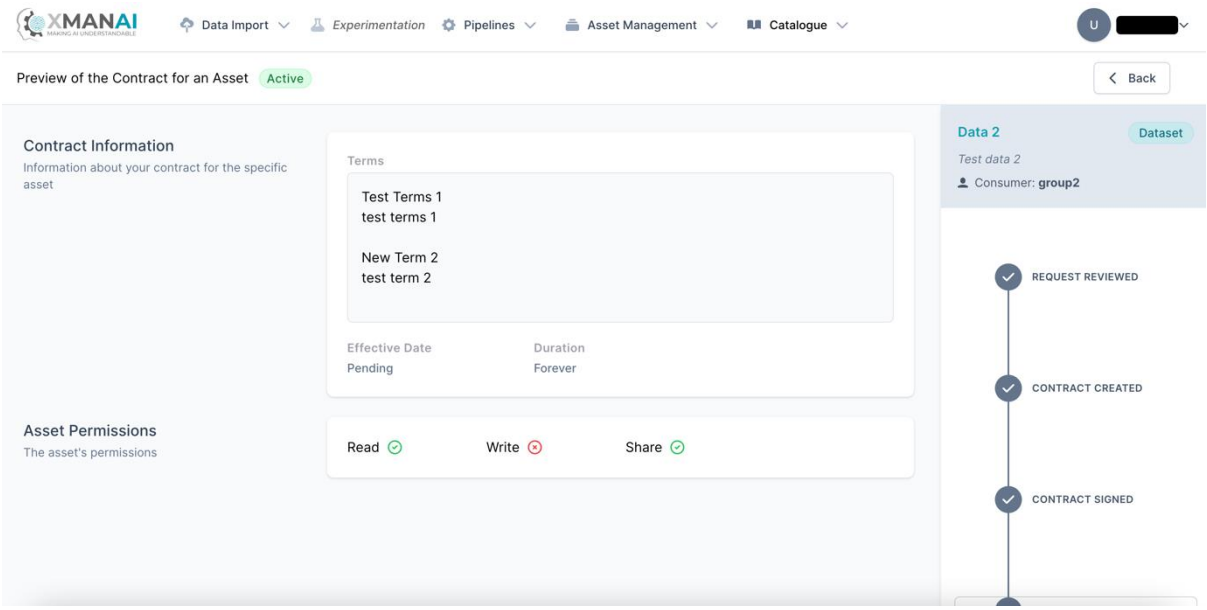


*Figure 2-32: XAI Marketplace - Contract Manager. View an active sharing agreement for an asset*

At any moment, a user may view the details of a contract including its terms, effective date and duration, as well as download it locally as a file for offline review.

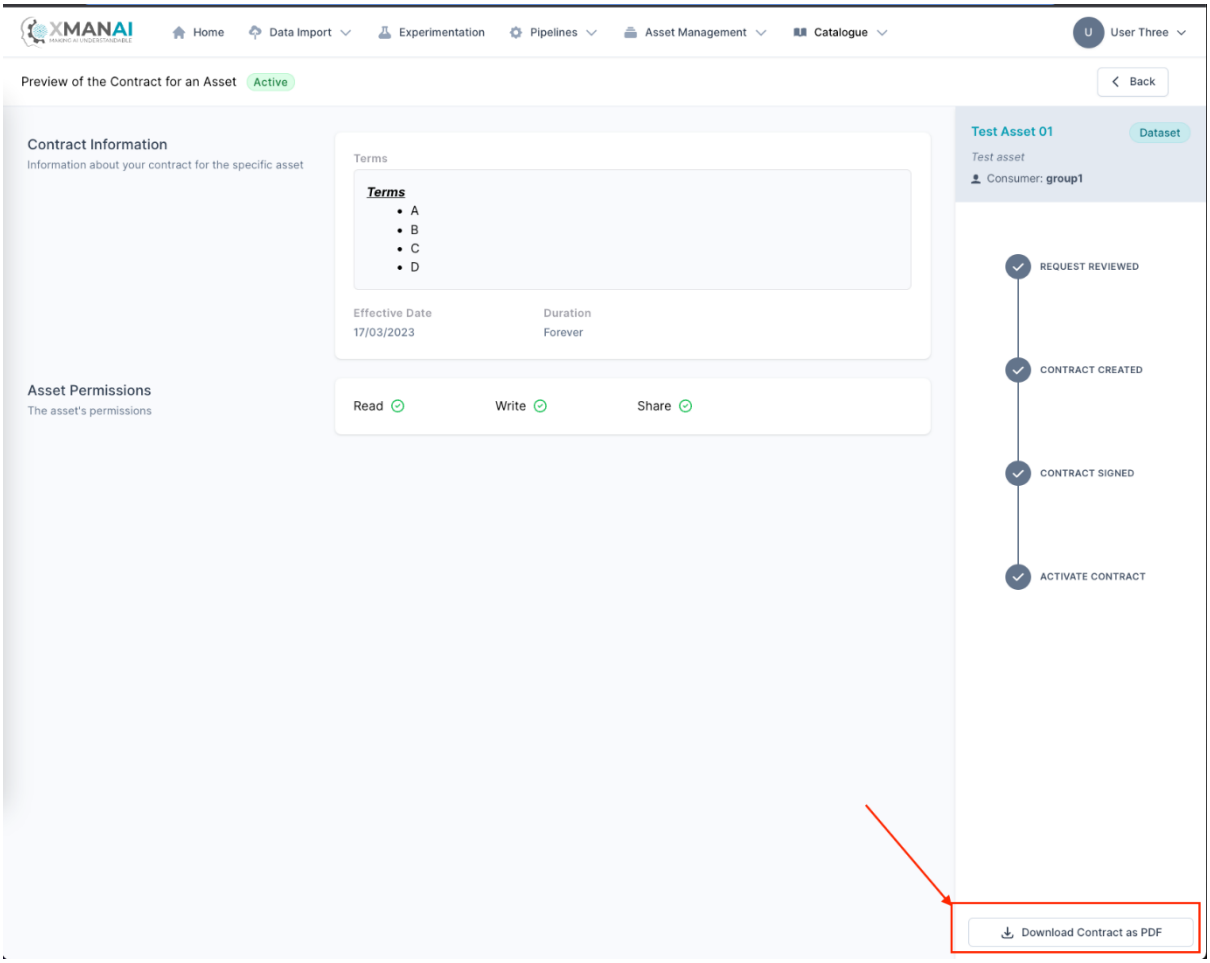


*Figure 2-33: XAI Marketplace - Contract Manager. View Contract Details*

### 2.8.4 Technology stack and License information

The Contract Manager component is based on the S5 Share Platform, which incorporates state-of-the-art technologies related to smart contracts and blockchain. More specifically, it employs the Ethereum distributed platform for the blockchain layer, along with its built-in smart contract functionalities. For the back-end layer, the web framework used is the Nest (NodeJS) and for the front-end layer the Vue.JS JavaScript framework. In XMANAI, in addition to the UI redesign, the smart contract terms and structure, including the asset permissions, as well as the end-to-end sharing process, have been updated. The Contract Manager is a proprietary software owned and developed by Suite5.

### 2.8.5 Code repository

The code of the component is published in a private section of the XMANAI repository in GitLab[18], access to which can be provided by a request sent to the project coordinator.

### 2.8.6 Improvements since the previous version

In the final release of the Contract Manager, a set of improvements have been introduced (in addition to bug fixes and UI improvements):

- The results of an XAI pipeline are added to the list of assets (i.e. datasets, AI models) that are considered as assets to be available for inclusion in a smart contract.
- The contract details can be exported and downloaded as a file.

### 2.8.7 Considerations

An asset that has type “result” is essentially the outcome of an XAI pipeline that may use various datasets and AI models as input, and there may be different/conflicting IPR among such assets. In order to streamline the sharing process though (based on the sharing scenarios that are expected within the XMANAI project and in view of the future exploitation activities), it was decided to handle the sharing of the “derivative” result as an independent outcome, irrespectively of its input assets.

In addition, at the time when D2.2 was written, experiments and XAI pipelines were also considered as AI assets that were within the scope of the Contract Manager. However, in the course of the integration activities, it was acknowledged that an experiment is always dependent on an AI model and thus is not to be shared on their own (as an independent asset) across organizations. In addition, XAI pipelines are to be shared within the same organization and not in a cross-organizational context due to the technical implications that would otherwise arise and the lack of any associated business needs.

Finally, taking into consideration that the integration activities are ongoing for the beta release of the XMANAI Platform on M32 and that the final version of all Services Bundles has been released at the same time that the alpha phase of the demonstration activities (D6.4) concluded, additional updates may occur (during the WP5 activities) in the Contract Manager to address any additional needs that may emerge.

## 2.9 Provenance Engine

### 2.9.1 Description

[18] https://GitLab.com/xmanai-h2020

The Provenance Engine collects, processes and manages provenance data in a W3C-PROV compliant manner. This includes retrieving provenance information like *who* performed *which CRUD-operation* involving *which version* of *which asset* from the Assets Store and storing the respective metadata as triplets in an RDF-triplestore. On request, information representing the lineage, operation history or status of an asset are provided in a structured manner.

*Table 2-15: Provenance Engine - Functionalities and Status*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| PE.1 | Offer endpoints to submit all CRUD operations and store the resulting metadata as triplets in a W3C-PROV compliant manner. (implemented) |
| PE.2 | Combine the available metadata and offer endpoints to retrieve them in a structured format, providing information about the lineage, operation history and status of an asset. (implemented) |

### 2.9.2 API documentation

A complete Swagger documentation of the API can be found in the project's code repository[19]. The following table additionally provides an overview of all endpoints.

*Table 2-16: Provenance Engine – API Endpoints Overview*

| Endpoint | Method | Description |
|---|---|---|
| /create_asset | POST | Create a provenance entry for the operation of a new asset being **created**.<br>Arguments:<br>• username<br>• uuid<br>• asset_name<br>• asset_type<br>• version_id |
| /update_asset | POST | Create a provenance entry for the **update** operation performed on an existing asset.<br>Arguments:<br>• username<br>• uuid<br>• asset_name<br>• asset_type<br>• version_id<br>• version_id_prev |

[19] https://gitlab.com/xmanai-h2020/api-docs/-/blob/main/Provenance Engine v1.yaml

| Endpoint | Method | Description |
|---|---|---|
| /read_asset | POST | Create a provenance entry for the **read** operation being performed on an existing asset.<br>Arguments:<br>• username<br>• uuid<br>• asset_name<br>• asset_type<br>• version_id |
| /delete_asset | POST | Create a provenance entry for the **delete** operation being performed on an existing asset.<br>Arguments:<br>• username<br>• uuid |
| /get_asset_family_tree | GET | Get complete family tree of specified asset.<br>Arguments:<br>• uuid<br>• asset_type |
| /get_asset_lineage | GET | Get a specified lineage of a specified asset.<br>Arguments:<br>• uuid<br>• asset_type<br>• lineage_id |
| /get_asset_history | GET | Get the complete operation history of a specified asset.<br>Arguments:<br>• uuid<br>• asset_type |
| /get_asset_status | GET | Get the last operation performed on a specified asset.<br>Arguments:<br>• uuid<br>• asset_type |

### 2.9.3 User Interface

The component doesn't possess a graphical user interface.

### 2.9.4 Technology stack and License information

The creation of W3C-prov compliant metadata in RDF-format is handled by Python prov[20], which is published under MIT license. Its services will be provided via a REST API built based on Python Flask[21], which is published under BSD3 clause license. The metadata themselves are being stored in a Virtuoso RDF Triple Store[22], which is published under GPL.

### 2.9.5 Code repository

The code of the component is published in a private section of the XMANAI repository in GitLab[23], access to which can be provided by a request sent to the project coordinator.

### 2.9.6 Improvements since the previous version

The previous release of the Industrial Asset Management and Secure Asset Sharing Bundles didn't include the Provenance Engine. Hence, all of the above-mentioned functionalities are to be considered as a progress compared to the previous version.

### 2.9.7 Considerations

There are no specific considerations at the moment.

## 2.10 Access Manager: Policy Engine

### 2.10.1 Description

The scope of the Policy Engine is to provide the robust and solid access control mechanism that facilitates the dynamic and effective regulation of the access to the various assets of the XMANAI platform. In this sense, the Policy Engine regulates the access to the underlying assets of the platform by formulating an access control decision for each request that is received. The basis of this mechanism are the access policies, as provided by the Policy Editor, and the composed access control model based on which the access control decisions are formulated.

The final version of the component is delivering all the planned functionalities of the component building on top of the previous initial version that was delivered and documented in deliverable D2.2, as well as the detailed design specifications that were documented in deliverable D2.1. As described in the previous versions of the deliverable, the Policy Engine is a purely backend component hence it

[20] https://github.com/trungdong/prov

[21] https://github.com/pallets/flask

[22] https://github.com/openlink/virtuoso-opensource

[23] https://GitLab.com/xmanai-h2020

provides a rich set of backend operations which are leveraged by the rest of the components via well-defined REST interfaces. The Policy Engine delivers two major sets of functionalities implemented as backend operations. The first set is focused on the formulation of the XMANAI access control model while the second set is related to the formulation of the access control decisions.

The generation of the access control model is performed based on the design specifications of the hybrid approach that is adopted in XMANAI, as described in deliverables D2.1 and D2.2 (**PEN1**). To this end, the Policy Engine receives and combines the latest access control policies, as provided by the Policy Editor, in order to formulate the solid access control model that is utilised for the evaluation of new access requests that are received. To meet its goals, the Policy Engine enables the formulation of more fine-grained access policies with the combination and enforcement of multiple complementary access policies based on logical reasoning (**PEN.3**). Hence, the optimised Policy Engine offers the functionalities which enable the constant deployment, update and enforcement of any new or updated access policy or a combination of access policies (**PEN.2**) which are provided by the Policy Editor.

Furthermore, this access control model is leveraged to formulate all the access control decisions. To achieve this, all access requests are evaluated against the defined access control model to yield an access control decision that will allow or deny the access to the requested asset (**PEN.4**). The particular set of functionalities is realised through a rich set of well-defined API interfaces which are leveraged by the rest of the components of the platform in order to receive an access control decision from the Policy Engine (**PEN.5**).

The main functionalities offered in the final release of the Policy Engine component are depicted in the following table:

*Table 2-17: Policy Engine – Implemented Functionalities*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| PEN1 | **Formulation of the access control model.** In the final release, the Policy Engine is able to formulate the access control model for the protected asset and to receive the access policies as defined by the asset owner via the Policy Editor. The final release is built directly on top of the previous and several optimisations were introduced towards the most flexible and effective formulation of the access control model taking into consideration the updates introduced on PEN2 and PEN3 as listed below. |
| PEN2 | **Instant deployment, update and enforcement of the access policies.** In the final release, the Policy Engine is able to handle the evolution of access policies by receiving and instantly deploying and enforcing the new or updated more fine-grained access policies as received by the Policy Editor. |
| PEN3 | **Combination and enforcement of multiple policies for an asset based on logical reasoning**. In the final release, the Policy Engine has been enhanced to support the definition of more fine-grained access policies where multiple conditions can be defined based on logical reasoning. |
| PEN4 | **Regulate the access to any asset of XMANAI**. In the final release, the Policy Engine supports the reception of an access control evaluation request and the formulation of the appropriate access control decision based on the current access control model. In particular, the advancements introduced in PEN1, PEN2 and PEN3 resulted into a series of enhancements that were introduced and incorporated into the Policy Engine's access control evaluation mechanism. |
| PEN5 | **Provide a set of well-defined API interfaces for the access control mechanism**. In the final release, the Policy Engine exposes an updated and fine-tuned set of APIs |

| | |
|---|---|
| | suitable for the reception and evaluation of all access requests. The provided APIs facilitate the optimal usage of the component by the rest of the components. |

### 2.10.2 API documentation

The Policy Engine provides a further extended and updated set of well-defined APIs taking into consideration the updated implementation of all the features mentioned in the previous paragraph. The provided APIs allow the consumption of the provided features by the rest of the components of the platform. The following table presents the core endpoints exposed by Policy Engine while the complete documentation of the API endpoints is also publicly available in the project code repository[24].

*Table 2-18: Policy Engine – API Endpoints Overview*

| Endpoint | Method | Description |
|---|---|---|
| /api/v1/asset-policy-engine/check-one | POST | Checks if the user has access to a specific asset. |
| /api/v1/asset-policy-engine/check-many | POST | Checks if the user has access to a list of assets. |
| /api/v1/asset-policy-engine/check-all | GET | Returns a list of all the assets that the user has access to. |

### 2.10.3 User Interface

The Policy Engine constitutes a purely backend component interacting with the rest of the components of the platform via well-defined REST interfaces, hence no user-interface is foreseen for this component.

### 2.10.4 Technology stack and License information

The implementation of the final version of the Policy Engine is based on a set of mature and powerful frameworks, libraries and technologies that have been successfully combined and integrated in order to realise the designed functionalities.

To this end, the list of frameworks, libraries and technologies utilised for the Policy Engine is composed of:

- Casbin[25], the powerful policy enforcement framework which provides a rich set of functionalities required for the implementation of the access control model.
- jCasbin[26], the Java-based library of Casbin which supports the implementation of the access control mechanism based on Casbin.
- Spring Boot framework[27], the dominant Java-based framework that enables the development of robust enterprise-ready applications with the widest set of plugins, libraries and extensions for the Java 17 programming language.

[24] https://policy-manager-backend.ai4manufacturing.eu/swagger-ui.html
[25] Casbin, https://casbin.org/
[26] jCasbin, https://github.com/casbin/jcasbin
[27] Spring Boot framework, https://spring.io/projects/spring-boot

- PostgreSQL[28], a powerful RDBMS offering ACID compliance and a large variety of features that are suitable for the needs of the Policy Engine's backend operations.

In terms of licensing, the Policy Engine component is closed source. The developed version is available through the XMANAI platform.

### 2.10.5 Code repository

As explained in the previous paragraph, the Policy Engine component is closed source. The code of the component is published in a private section of the XMANAI repository in GitLab[29], access to which can be provided by a request sent to the project coordinator.

### 2.10.6 Improvements since the previous version

In the final version of the Policy Engine, a set of improvements and enhancements were introduced on all implemented features. In particular, a more flexible and efficient process for the formulation of the access control model has been introduced. This was mandatory as the received access policies were optimised to be even more fine-grained and their combination was further enhanced also. The updates on the access policies also introduced updates to the already established mechanism that receives, updates and enforces these policies into the access control model. The access control mechanism that leverages the access control model was also optimised to incorporate the aforementioned update, while also the API endpoints exposed by the component were updated as a result of all the propagated changes in order to expose the new capabilities of the Policy Engine.

### 2.10.7 Considerations

No challenges or limitations were faced during the development of the final version of the Policy Engine.

## 2.11 Access Manager: Policy Editor

### 2.11.1 Description

The scope of the Policy Editor is to undertake the complete access policy lifecycle management, spanning from the creation and storage of an access policy to the reuse, update and deletion of the access policy. As explained also in section 2.11, an access policy constitutes a core of the overall XMANAI access control mechanism as it defines the exact authorisation rules which regulate the access to any asset of the platform and are explicitly set by the owner of each asset. To meet its goals, the Policy Editor in its final version also offers a novel and user-friendly user-interface through which the owner of the asset is able to perform all the access policy lifecycle management operations.

Based on the design specifications that were documented in deliverable D2.1 and the first version of the Policy Editor that was delivered with deliverable D2.2, the final version builds on top of the previous version in order to offer a rich set of features that can be conceptually organised into: a) the solid access policy lifecycle management operations and b) the respective user-interface and the interaction with the Policy Engine in order to provide the required input for the formulation of the

[28] PostgreSQL, https://www.postgresql.org/
[29] https://GitLab.com/xmanai-h2020

access control model. To this end, this final version of the component delivers the complete toolset for the definition, storage, update and deletion of access policies for any protected asset of the XMANAI platform with the respective enhanced backend (**PED.1**) and frontend functionalities (**PED.4**). In addition to this, the Policy Editor facilitates the further reuse of existing policies on a different asset (**PED.2**) via the newly introduced operations, the definition of fine-grained access policies where a single condition is set, as well as more complex access policies where multiple conditions can be defined based on logical reasoning (**PED.3**). On the other hand, the Policy Editor undertakes the responsibility of constantly propagating the newly created or updated access policies to the Policy Engine (**PED.6**) adhering to the syntactic and logical rules of the fine-grained policies and advanced restrictions imposed by the access control model's implementation as set by the Policy Engine (**PED.5**).

The final release of the Policy Editor implements the complete set of features that were documented in detail in the deliverable D2.1. The following table provides an overview of the final status of the implementation in this final release.

*Table 2-19: Policy Editor – Implemented Functionalities*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| PED.1 | **Enable the flexible definition, storage, update and deletion of access policies.** In this final release, the Policy Editor facilitates all the CRUD operations for more fine-grained access policies. Furthermore, in this final release the support for protected assets was further extended to support newly introduced assets (for example the assets produced by the Model Guard component) that derived from the evolution of the XMANAI platform. |
| PED.2 | **Facilitate the reuse of previous defined access policies.** In this final release, the Policy Editor enables the reuse of existing policies for a different asset of the same owner through this intuitive user interface. |
| PED.3 | **Enable the combination of the multiple access policies using Boolean logic**. In this final release, the Policy Editor provides the means to define complex access policies with a combination of multiple conditions with use of logical reasoning taking into consideration the advancements implemented for PED.1 and PED.2. |
| PED.4 | **Provide the easy-to-use and novel user-interface for all the access policy lifecycle management operations**. This final version of the component delivers the easy to use and intuitive user interface where all the access policy lifecycle operations are performed by the asset owner. |
| PED.5 | **Enable the loading of the access policies into the Policy Engine.** In this final release, the Policy Editor facilitates the loading of the defined fine-grained access policies for all the available assets of the platform into the Policy Engine, undertaking the integration aspects that trigger the access control model definition or update. |
| PED.6 | **Immediate propagation of newly created or updated access policies.** In this final release, the Policy Editor facilitates the immediate update of the access control model by interacting with the Policy Engine, by incorporating all the required changes and updates that were introduced in the Policy Engine as described in section 2.11. |

### 2.11.2 API documentation

In the final version of the Policy Editor an updated set of well-defined APIs that interconnect the backend and frontend operations of the component is delivered. The following table presents the core endpoints exposed by Policy Editor while the complete documentation of the API endpoints is also publicly available in the following URL:

https://policy-manager-backend.ai4manufacturing.eu/swagger-ui.html

*Table 2-20: Policy Editor – API Endpoints Overview*

| Endpoint | Method | Description |
|---|---|---|
| /api/v1/asset-policy-editor | POST | Adds a new set of rules for a specific asset. |
| /api/v1/asset-policy-editor | GET | Returns the existing rules for a specific asset. |
| /api/v1/asset-policy-editor | PUT | Updates the existing rules for a specific asset. |
| /api/v1/asset-policy-editor | DELETE | Deletes the existing rules for a specific asset. |
| /api/v1/asset-policy-editor/reusable | GET | Returns all the policies that can be reused. |

### 2.11.3 User Interface

In the final release of the Policy Editor, the user-friendly user interface where all the access policy lifecycle operations are performed by the asset owner is delivered. In particular, the user is able to define or edit the data access policies that are enforced on the specific dataset by accessing the relevant page provided by the Policy Editor. Initially the user selects the applied access type where *Public* means no access control is applied, *Confidential* means that only the users of the asset's owner's organisation can access the asset and *Restricted* means that access policies are defined (Figure 2-34).

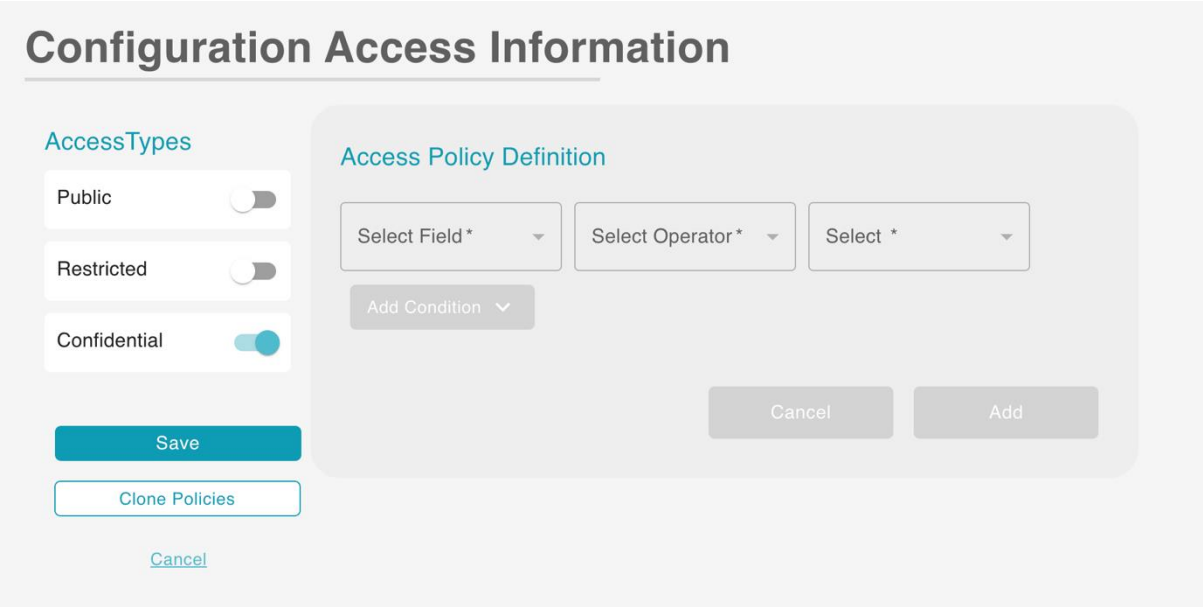


*Figure 2-34: Policy Editor – Setting the desired Access Type*

In this page, the user can provide either a single rule or combination of rules based on a number of attributes of the candidate requestor. In the latter case, the attributes are logically combined in order to define more complex and fine-grained access policies (Figure 2-35). Each access policy is defined using the “equals” or “does not equal” conditions as well as the respective values. This is translated into an access policy that defines whether the access should be granted to the requestor or not.

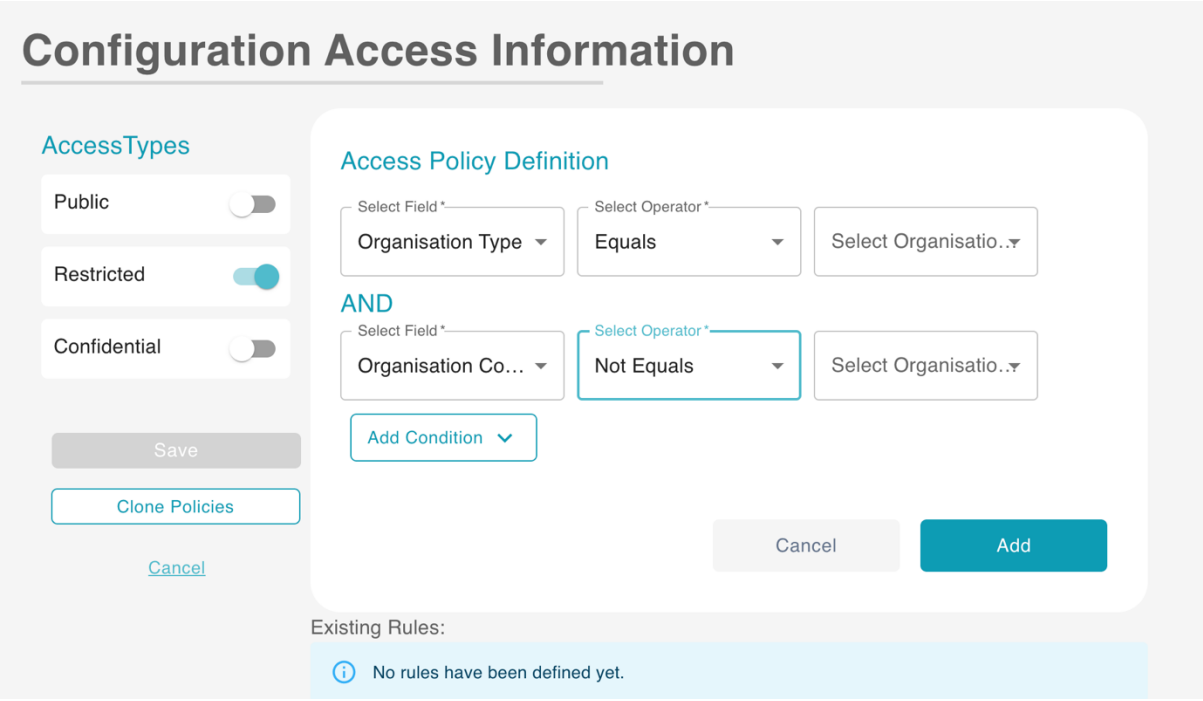


*Figure 2-35: Policy Editor – Definition of complex access policies*

The user is also able to edit (by pressing the pencil icon) or delete (by pressing the trash bin icon) any existing rule that is currently applied on the specific dataset as listed in the Existing Rules list or clone the policies from another asset that he/she owns (Figure 2-36).

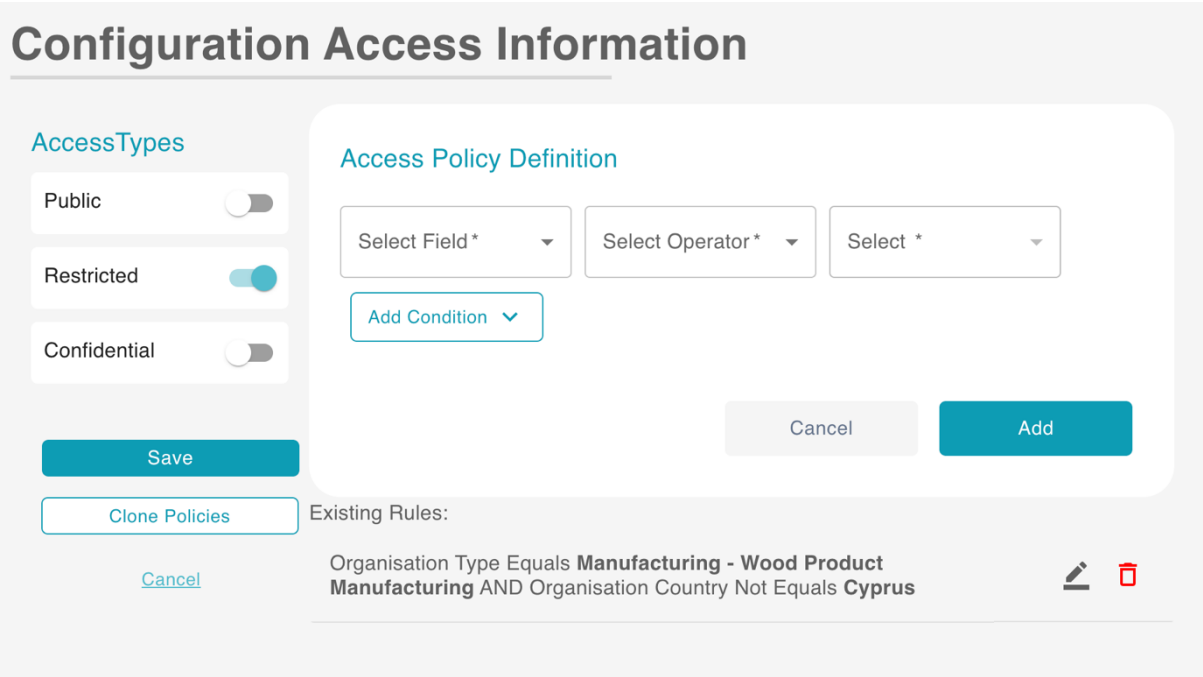


*Figure 2-36: Policy Editor – Edit or delete existing access policies*

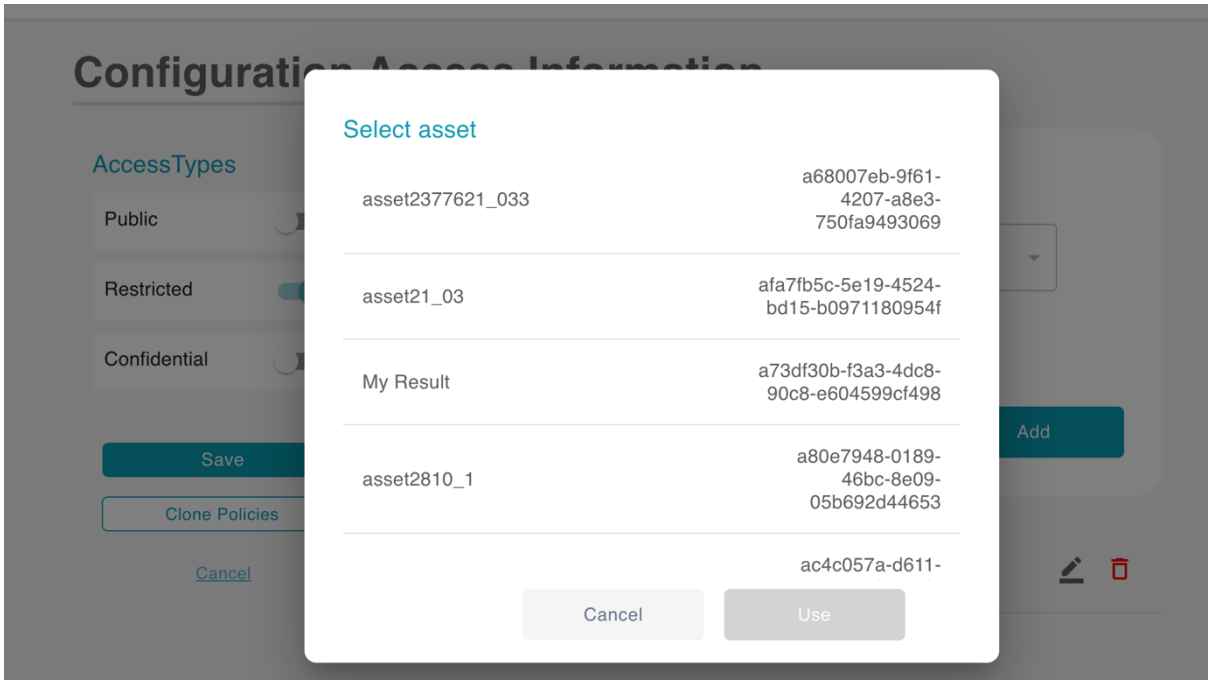


*Figure 2-37: Policy Editor – Clone access policies*

## 2.11.4 Technology stack and License information

The implementation of the final version of the Policy Editor is based on the same list of frameworks, libraries and technologies which are leveraged for the Policy Engine. Hence, the technology stack of the Policy Editor includes Casbin, jCasbin and Spring Boot framework for the backend operations of the component. In addition to this, the well-known RDBMS solution of PostgreSQL is utilised for the storage of the access policies.

The frontend operations are based on React, the dominant open-source JavaScript framework, which is one of the most powerful and commonly used Single Page Applications frameworks. React provides a rich set of functionalities for novel frontend application development.

In terms of licensing, the Policy Engine component is closed source. The developed version is available through the XMANAI platform.

## 2.11.5 Code repository

As explained in the previous paragraph, the Policy Engine component is closed source. The code of the component is published in a private section of the XMANAI repository in GitLab[30], access to which can be provided by a request sent to the project coordinator.

## 2.11.6 Improvements since the previous version

The major update of the final release is the delivery of the novel user interface of the Policy Editor which enables the execution of the access policy lifecycle operations (including their reuse and cloning) by the user as presented in section 2.11.3. In addition to this, the final version has been

[30] https://GitLab.com/xmanai-h2020

optimised in order to incorporate the updates introduced in the access policies as well as all related CRUD performed on them.

### 2.11.7 Considerations

No challenges or limitations were faced during the development of the final version of the Policy Editor.

## 2.12 Identity & Authorisation Management

### 2.12.1 Description

The scope of the Identity and Authorisation Manager is two-fold. On the one hand, it provides the holistic user account management lifecycle of the XMANAI platform, offering the single core identity provider of the platform that provides the registration, verification and authentication operations. It undertakes the whole registration process as well as a user invitation process utilising the concept of the organisations in order to group the users of the platform. On the other hand, it provides the authorisation mechanism which regulates the intercommunication of the various layers or components of the platform. The Identity and Authorisation Manager controls the intercommunication between the components based on authorisation rules, serving the role of the mediator for all requests.

The final version of the Identity and Authorisation Manager incorporates all the features that were designed and documented in deliverable D2.1 and the advancements that were introduced on top of the initial release that was delivered with deliverable D2.2. In particular, the Identity and Authorisation Manager provides all the enhanced functionalities related to the user management of the platform, starting from the registration and invitation of an organization to the maintenance operations of the organisation's profile (**IAM.01**). In addition to this, the Identity and Authorisation Manager provides all the updated operations for the management of the organisation's users, covering all the aspects from their invitation and registration to their profile update, suspension and deletion (**IAM.02**). Besides the user management operations, it provides the optimised authentication mechanism that verifies and controls the access to the platform's services and offerings with a secure login mechanism (**IAM.03**). Finally, the Identity and Authorisation Manager undertakes the regulation of the intercommunication between the components with an authorisation mechanism at a service level or an endpoint level for each component (**IAM.04**).

The Identity and Authorisation Manager constitutes a core part of the backbone of the XMANAI platform. The first release delivered the complete set of features that were documented in detail in the deliverable D2.1 to facilitate the implementation of the rest of the components. However, in this final release several optimisations were introduced in order to include all the advancements of the platform. The following table provides an overview of the final status of the implementation in the current release.

*Table 2-21: Identity & Authorisation Management – Implemented Functionalities*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| IAM.01 | **Complete user management lifecycle by implementing solid organization registration**. In the final release of the Identity and Authorisation Manager the implemented mechanism for the registration and verification of an organisation, as well as the update of the organisation's profile have been further enhanced to be more user friendly and robust. In addition to this, the relevant user interface for the aforementioned operations has been delivered. |

| Feature ID | Functionalities & Implementation Status |
|---|---|
| IAM.02 | **Enable the invitation and registration of users under a single organization**. In the final release of the Identity and Authorisation Manager the implemented user's invitation and registration process as well as all the CRUD operations on the user's profile have been enhanced and the email mechanism has been introduced along with new email templates. in addition to this, the relevant user interface for the aforementioned operations has been delivered. |
| IAM.03 | **The authentication mechanism for the platform's services and offerings**. The final release incorporates the updated login mechanism that verifies and controls the access and that verifies the provided credentials is provided. Additionally, the support for token exchange and long running tokens has been added. |
| IAM.04 | **Regulate the intercommunication of the various components of the platform**. In this final release, the Identity and Authorisation Manager implemented the enhanced authorisation mechanism which controls the intercommunication level of the components on either a service or an endpoint level. |

### 2.12.2 API documentation

The final version of the Identity and Authorisation Manager provides an extended set of well-defined APIs, as explained in the previous section, that allow the consumption of the provided features by the rest of the components of the platform. The following table presents the core endpoints exposed by the Identity and Authorisation Manager while the complete documentation of the API endpoints is also publicly available in the following URL:

https://iam-backend.ai4manufacturing.eu/swagger-ui.html

*Table 2-22: Identity & Authorisation Management – API Endpoints Overview*

| Endpoint | Method | Description |
|---|---|---|
| /api/v1/organisation | POST | Request for a new organization in the platform. |
| /api/v1/organisation | GET | Show a list of all the organization registration requests. |
| /api/v1/organisation/{id}/approve | PATCH | Approve an organization registration request. |
| /api/v1/organization/{id }/decline | PATCH | Decline an organization registration request. |
| /api/v1/user-invitation | POST | Invite a new user to the platform. |

| Endpoint | Method | Description |
|---|---|---|
| /api/v1/user | POST | Finalize user registration to the platform, based on an existing invitation. |
| /api/v1/auth/login | POST | User login operation. Upon successful login the user receives back a JWT. |
| /api/v1/auth/verify | GET | Verify the identity of the user / check if user has a valid JWT. |
| /api/v1/account | GET | Return user's account details. |
| /api/v1/group/mine | GET | Return the organization that the user belongs to. |

### 2.12.3 User Interface

In this final version of the Identity and Authorisation Manager, the user interface that is leveraged by the users of the platform is delivered. The operations can be grouped into: a) the organisation management, b) the user management, c) user invitation, the login and forgot password forms and finally d) the service-level authorisation mechanism.

The Identity and Authorisation Manager offers the ability to the organisation's manager to perform the organisation's registration by completing the relevant form (Figure 2-38). Upon the approval of one of the platform's administrators, the organisation's manager is able to review, edit and update the details of the organisation (Figure 2-39), to invite new users as well as to view and edit the pending user invitations(Figure 2-40).

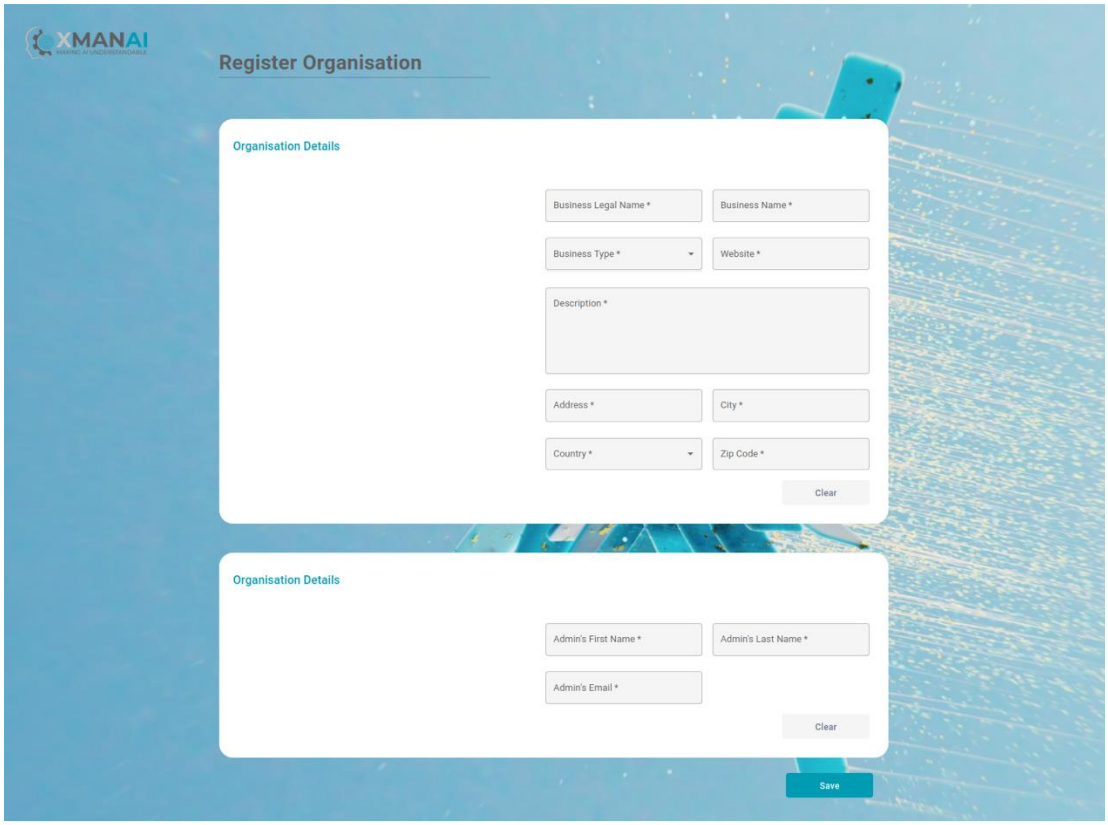


*Figure 2-38: Identity and Authorisation Manager – Organisation registration form.*

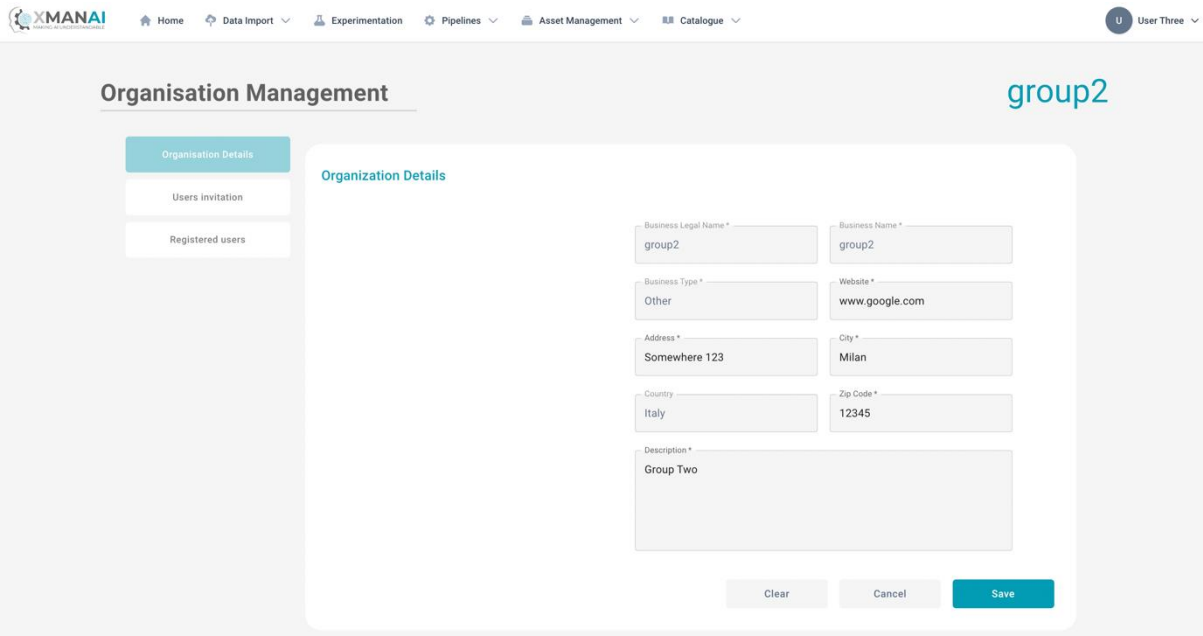


*Figure 2-39: **Identity and Authorisation Manager – Organisation details editing.***

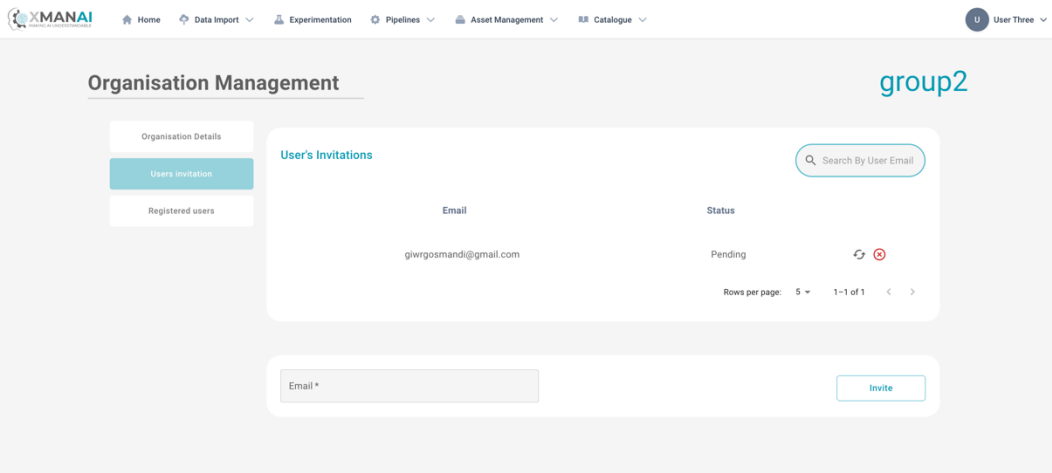


*Figure 2-40: Identity and Authorisation Manager – Users invitation.*

In addition to this, the organisation's manager is able to view the existing users of the organisation and edit them by deactivating/reactivating their accounts via the dedicated switch buttons.

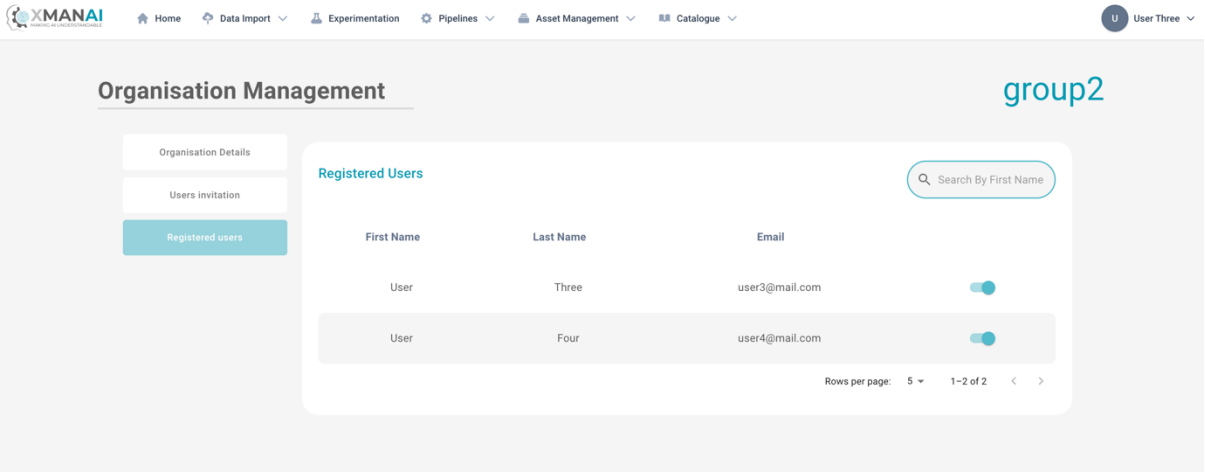


*Figure 2-41: Identity and Authorisation Manager – Registered users handling.*

Once a new user is invited by the organisation's manager, a new email is sent to his/her mailbox informing him/her to complete the pending registration to the platform (Figure 2-42). The user is provided with a link to complete the registration form that will give him/her access to the XMANAI platform (Figure 2-43). All users of the XMANAI platform can view and edit their personal profile details as well as change their passwords (Figure 2-44).

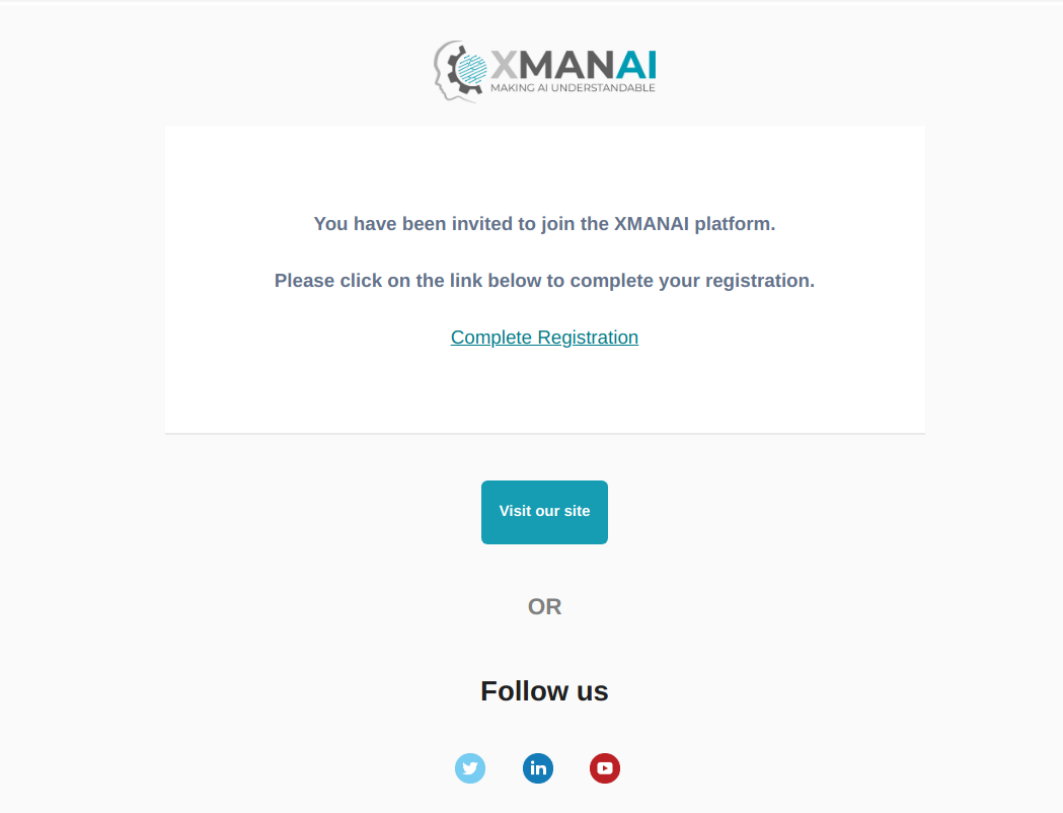


*Figure 2-42: XMANAI user registration invitation email*

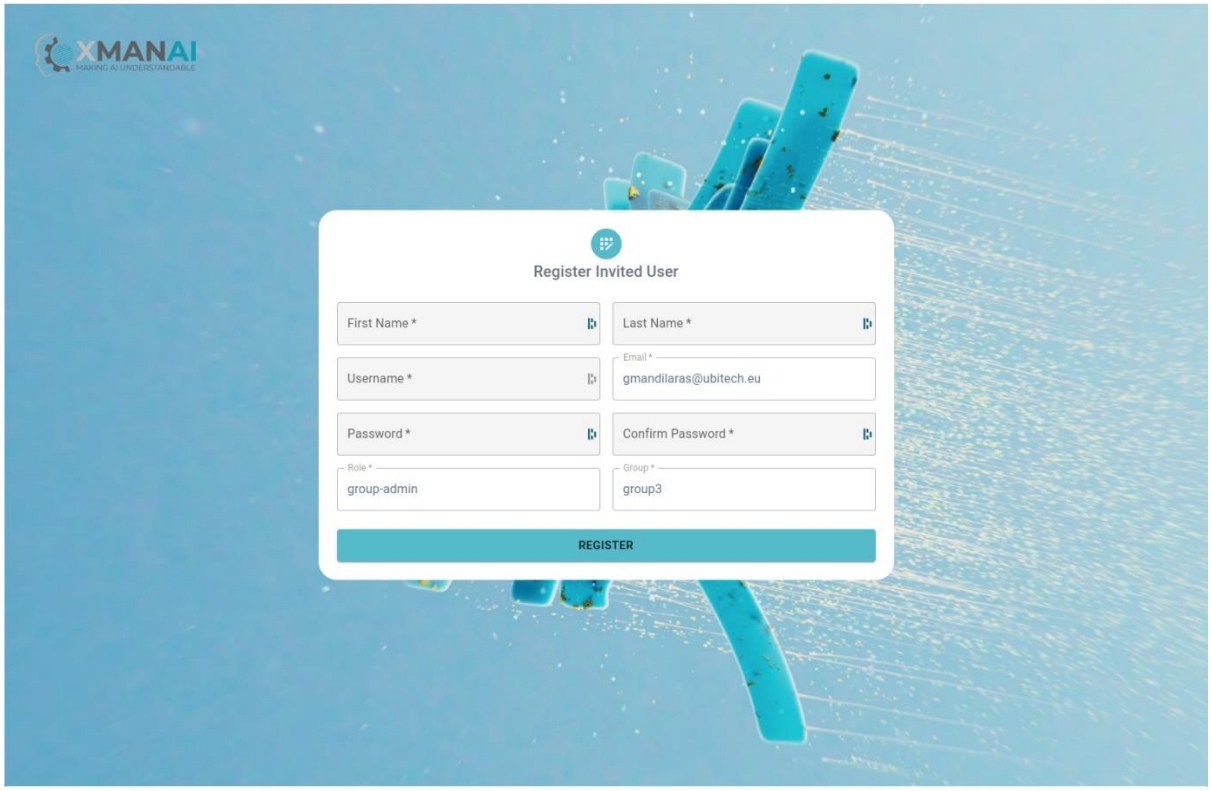


*Figure 2-43: Identity and Authorisation Manager – User registration form.*

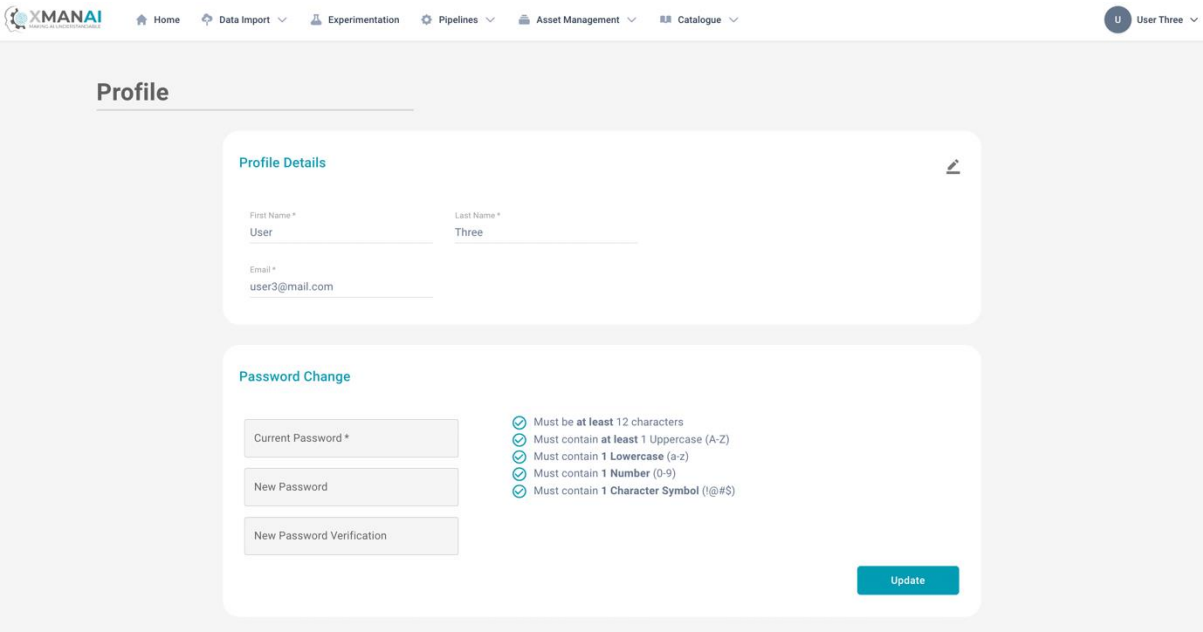


*Figure 2-44: Identity and Authorisation Manager – User profile.*

Access to the functionalities of the XMANAI platform is only made available to registered users upon providing their correct credentials. Hence, the user should firstly fill-in the login form in order to be successfully authenticated (Figure 2-45). In addition to this, the users are provided with the option to reset their password (Figure 2-46).

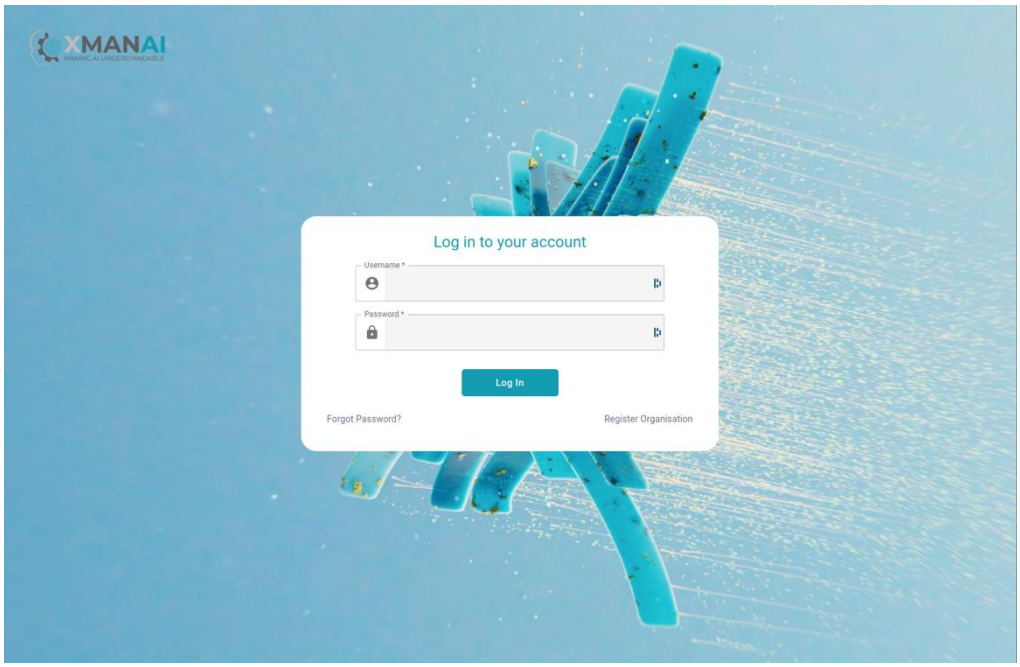


*Figure 2-45: Identity and Authorisation Manager – Login form.*

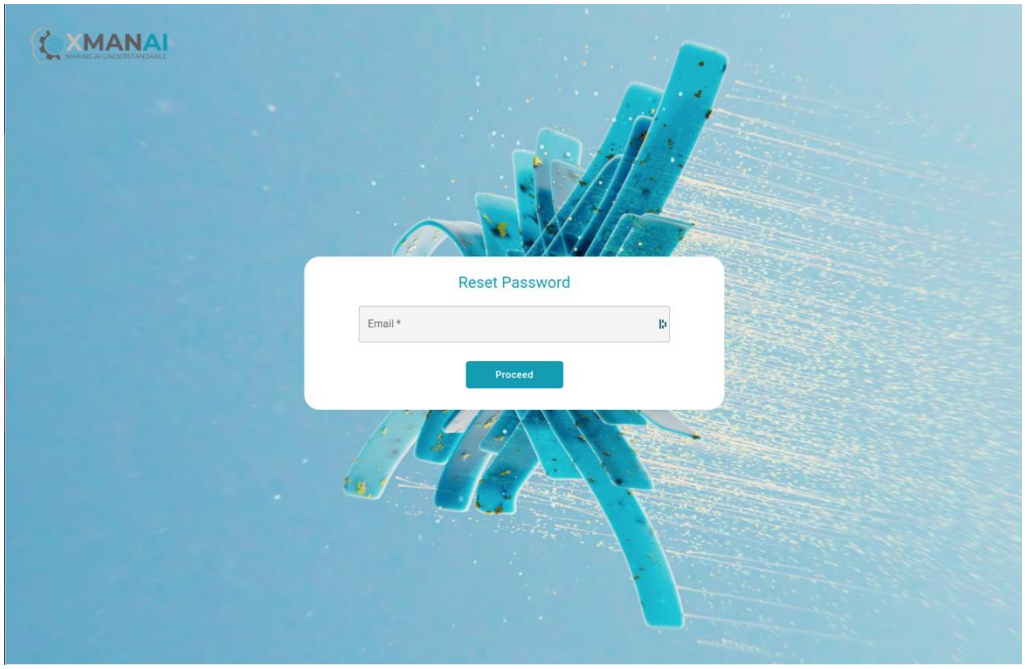


*Figure 2-46: Identity and Authorisation Manager – Forgot password form.*

The administrators of the platform are provided with a set of functionalities in order to properly manage the whole platform. In particular, the administrators are presented with the list of registration requests where they can either approve or reject any pending requests (Figure 2-47). Furthermore, the administrators are given access to all the information of the registered organisations and are provided with all the options offered on each organisation's manager such as to edit the organisation's details, manage user invitations and existing users and more (Figure 2-48, Figure 2-49, Figure 2-50).

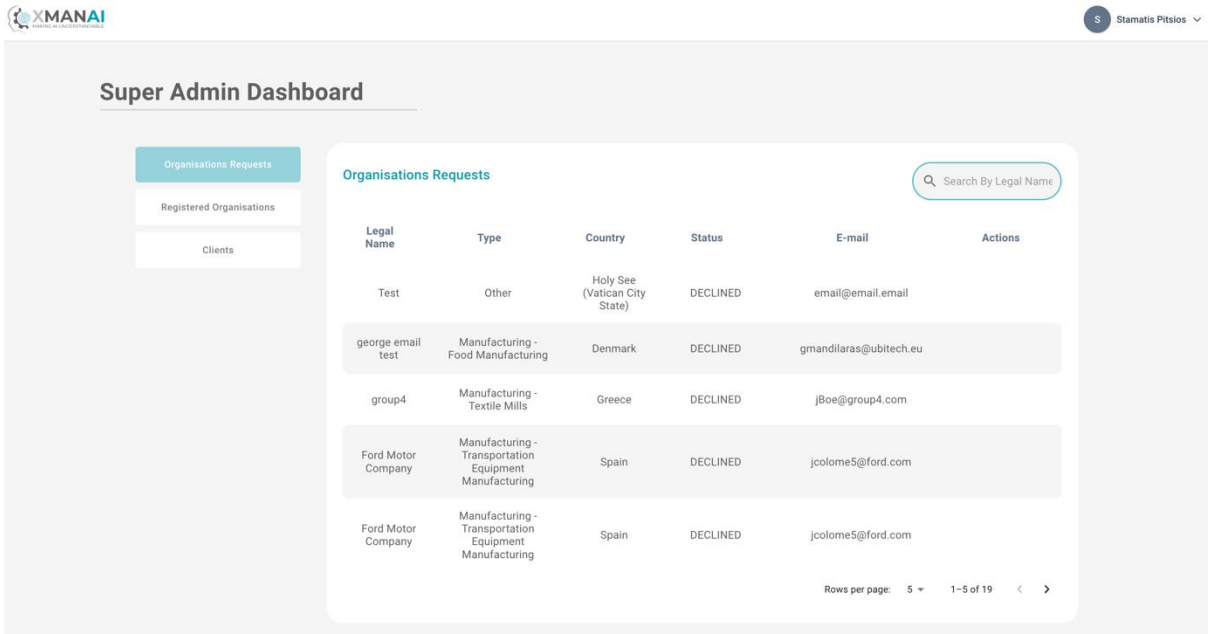


*Figure 2-47: Identity and Authorisation Manager – administration of organisation requests.*

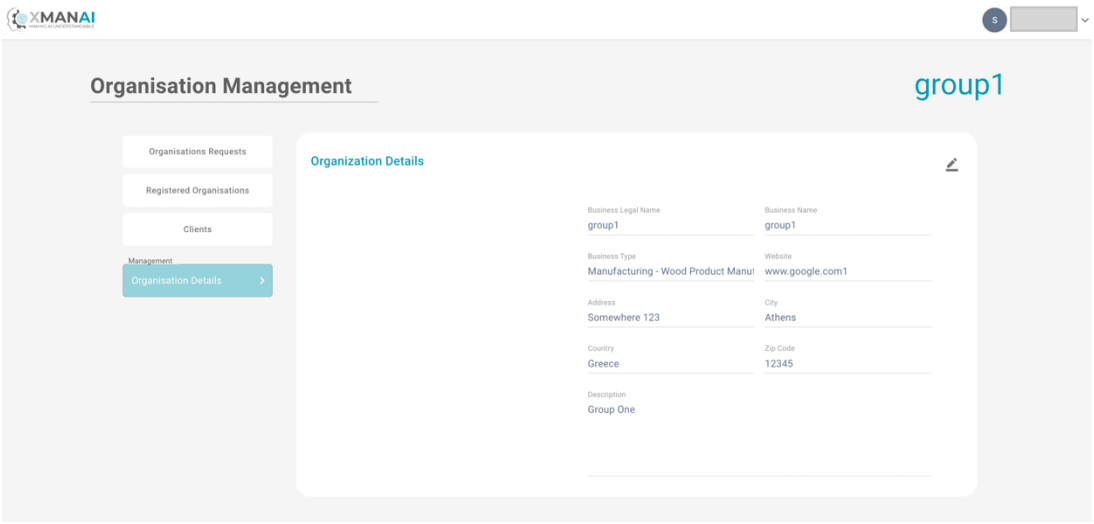


*Figure 2-48: Identity and Authorisation Manager – administration of organisation's details.*

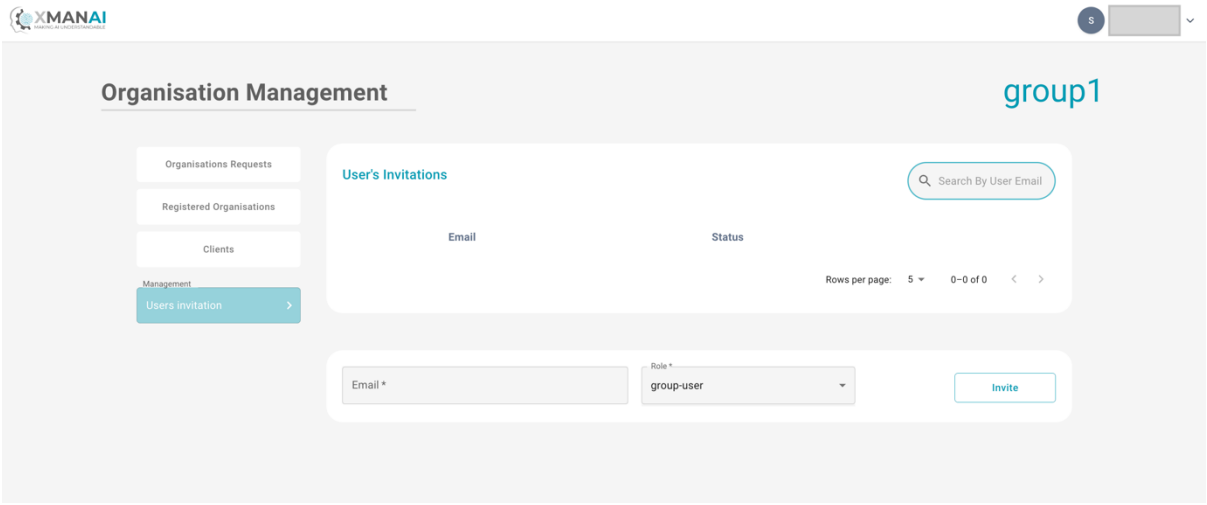


*Figure 2-49: Identity and Authorisation Manager – administration of organisation's user invitations.*

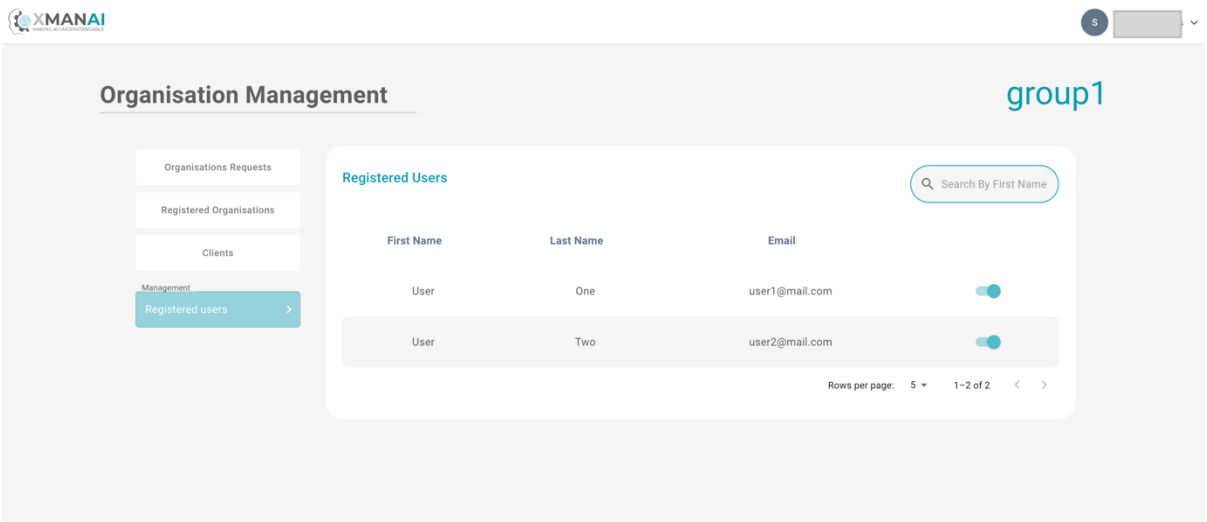


*Figure 2-50: Identity and Authorisation Manager – administration of organisation's users.*

Finally, the administrators of the platform are able to regulate the intercommunication of the various components of the platform via the provided service-level authorisation mechanism. In detail, the administrators are able to configure a service (Figure 2-51) and regulate the access to their resources by adding or editing the relevant information for each resource that should be properly protected (Figure 2-52).

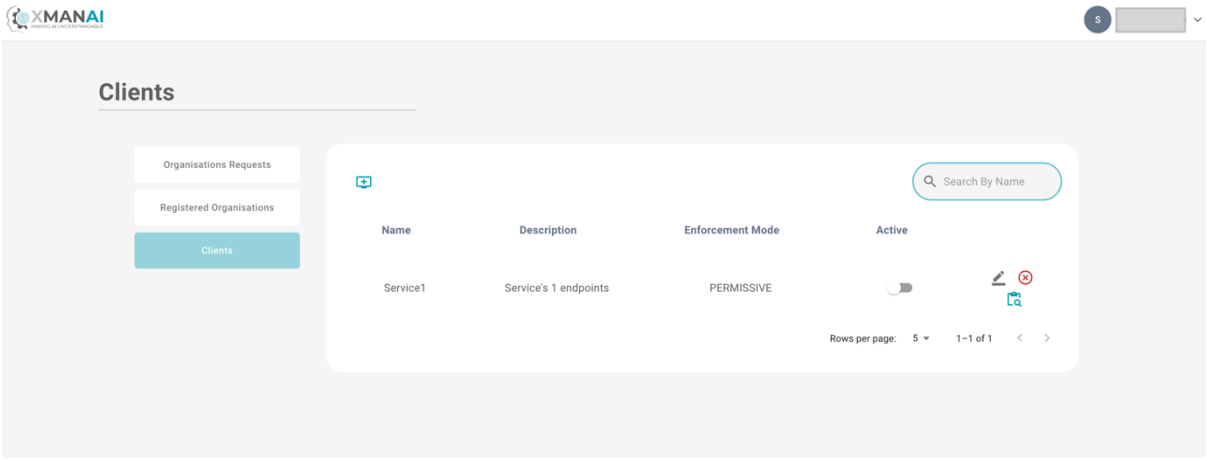


*Figure 2-51: Identity and Authorisation Manager – service-level authorisation.*

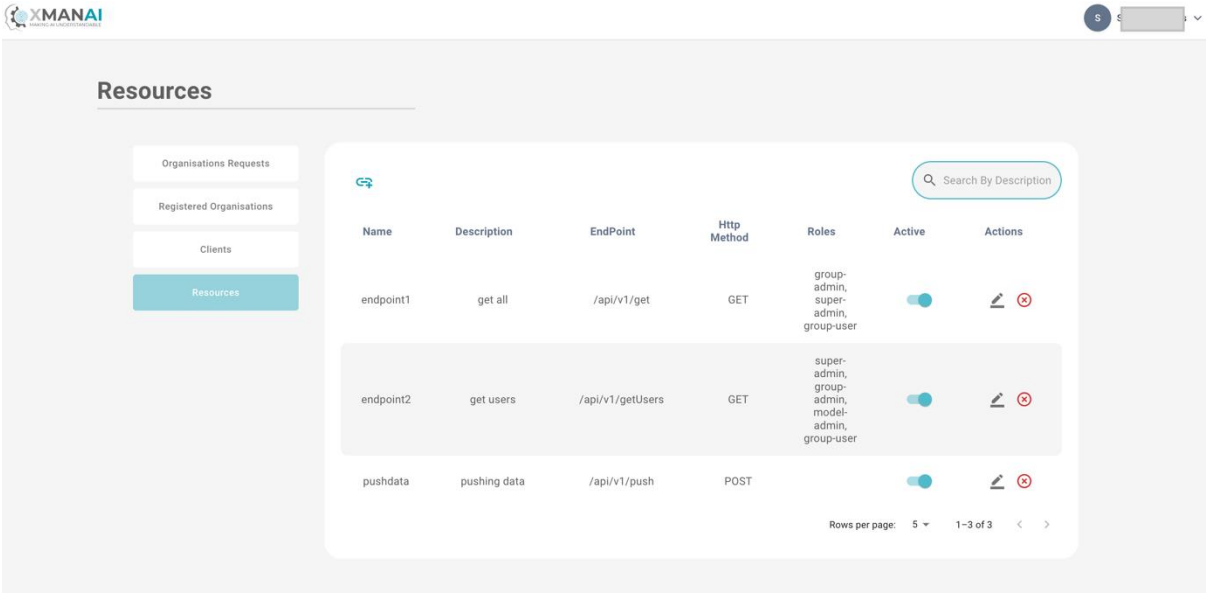


*Figure 2-52: Identity and Authorisation Manager – service's resources authorisation.*

## 2.12.4 Technology stack and License information

The implementation of the final version of the Identity and Authorisation Manager is based on a set of mature and powerful frameworks, libraries and technologies that have been successfully combined and integrated in order to realise the designed functionalities.

To this end, the list of frameworks, libraries and technologies utilised for the Identity and Authorisation Manager is composed of:

- Keycloak[31], the well-known identity and access management framework for robust identity management, authentication and access management operations. It provides a rich set of functionalities for the complete user management lifecycle.
- Spring WebFlux 3[32], the well-established reactive-stack web framework of Spring which is suitable for non-blocking APIs and web stack solutions.
- Spring OAuth2[33], the well-established powerful and highly customizable authentication and access-control framework of Spring.
- R2DBC[34], the most suitable database driver for Reactive designed from the ground up for reactive programming with SQL databases.
- PostgreSQL, a powerful RDBMS offering ACID compliance and a large variety of features that are suitable for the needs of the Results Visualisation Engine's backend operations.
- Redis[35], the widely used key-value store that provides fast processing and easy data structure representation.
- MinIO[36], the well-established high performance object storage solution that facilitates the building of high-performance infrastructure for machine learning, analytics and application data workloads.
- React[37], the dominant open-source JavaScript framework, which is one of the most powerful and commonly used Single Page Applications frameworks. React provides a rich set of functionalities for novel frontend application development.

In terms of licensing, the Identity and Authorisation Manager component is closed source. The developed version will be available through the XMANAI platform.

### 2.12.5 Code repository

As explained in the previous paragraph, the Identity and Authorisation Manager component is closed source. The code of the component is published in a private section of the XMANAI repository in GitLab[38], access to which can be provided by a request sent to the project coordinator.

### 2.12.6 Improvements since the previous version

The final release of the Identity and Authorisation Manager implements the user interface of the component (as presented in section 2.12.3) that supplements and supports a complete set of features that were also optimised and updated in order to incorporate the advancements of the platform. It incorporates updated user management operations as well as user invitation and registration processes with a new email mechanism and email templates. Additionally, it includes the support for token exchange and long running tokens while providing an enhanced service level authorisation process.

### 2.12.7 Considerations

[31] Keycloak, https://www.keycloak.org/
[32] Spring WebFlux , https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html
[33] Spring OAuth2, https://docs.spring.io/spring-security/reference/servlet/oauth2/index.html
[34] R2DBC, https://r2dbc.io/
[35] Redis, https://redis.io/
[36] MinIO, https://min.io/
[37] React, https://reactjs.org/
[38] https://GitLab.com/xmanai-h2020

No challenges or limitations were faced during the development of the final version of the Identity and Authorisation Manager.

## 2.13 Anonymiser

### 2.13.1 Description

The Data Anonymiser is a standalone component responsible for anonymizing a dataset before uploading it to XMANAI. We will use, as Anonymiser, the open-source tool Amnesia[39], which the ATHENA Research Centre developed in a previous research project. Amnesia can be downloaded locally as a desktop application. It is compliant with Microsoft and Linux operating systems. With Amnesia, the user can anonymize a dataset in four simple steps:

1. Import the original data

The original dataset must be in a simple text file with any delimiter. Amnesia has an import wizard, which guesses the data type and asks the user to confirm it. The user selects which fields will participate in the process and which will be left out.

2. Create the generalization hierarchies

The basic idea behind anonymizing a dataset is to replace unique values or combinations of values, e.g., zip code and date of birth, with more abstract ones, so they are no longer identified. Amnesia allows a user to create these rules for generalizing values in a semi-automatic way, to save them and re-use them or import them from other sources.

3. Apply an anonymization algorithm

The user can select the most suitable method for this problem (e.g., k-anonymity or km-anonymity) and link the hierarchies with the respective attributes of the records. The anonymization process starts.

4. Choose the solution you like

Amnesia depicts several possible solutions, visualizes the distribution of values, and provides statistics about the data quality in the anonymous dataset. A solution tailored to the user's needs can be inspected and applied with simple clicks. The anonymized data can then be saved locally.

*Table 2-23: Anonymiser – Available Functionalities*

| Feature ID | Functionalities & Implementation Status |
|---|---|
| AN.1 | Anonymise a tabular-form dataset.<br>This functionality is fully supported in the current development stage of the component. Amnesia provides full support for anonymizing a dataset in tabular format. |

### 2.13.2 API documentation

The Data Anonymiser is a standalone service that does not interact with other components. Therefore, it does not expose any API end-point.

### 2.13.3 User Interface

In this section, we illustrate some interfaces of the Data Anonymizer.

[39] https://amnesia.openaire.eu/index.html

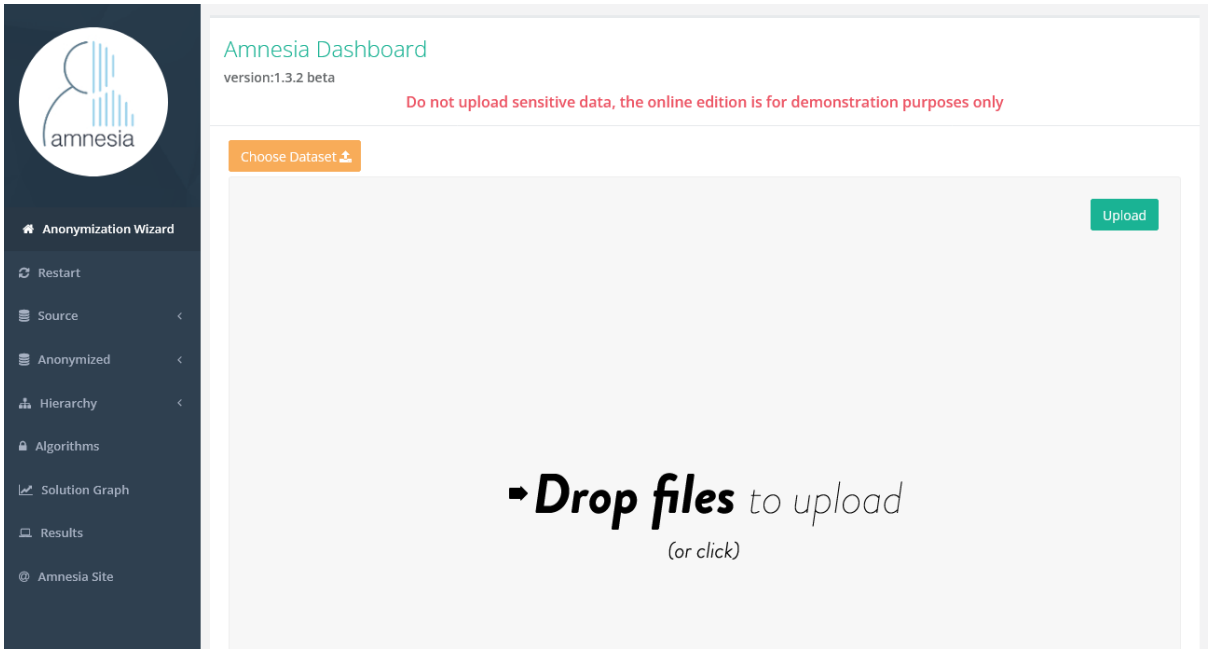


*Figure 2-53: The interface for uploading a tabular dataset in order to anonymize it.*

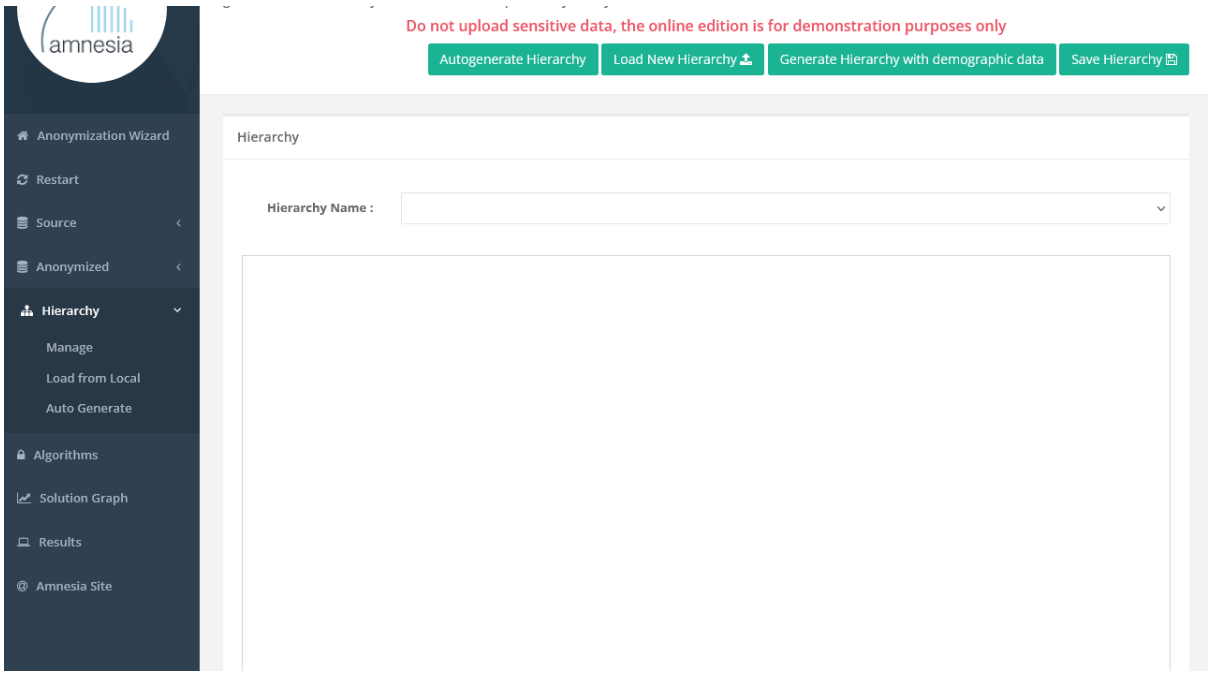


*Figure 2-54: The interface for generating a hierarchy, before applying an anonymization algorithm.*


*Figure 2-55: The interface for applying an anonymization algorithm.*

## 2.13.4 Technology stack and License information

Amnesia is developed with Java. Therefore, it requires a current Oracle or OpenJDK Java Runtime Environment with Java version 8 or greater.

## 2.13.5 Code repository

The source code for the Amnesia tool is managed in the GitHub repository[40].

## 2.13.6 Improvements since the previous version

Since Amnesia is a ready-to-use tool that is not developed as part of the XMANAI, there are no changes since the previous version.

## 2.13.7 Considerations

For the time being, there are no open issues to consider regarding the Anonymiser.

[40] https://github.com/dTsitsigkos/Amnesia

# 3 Conclusions and Next Steps

The current deliverable D2.3 “XMANAI Asset Management Bundles – Second Release” presents the second release of the components responsible in XMANAI for assets management. The design of their first version was presented in the deliverable D2.1 “Asset Management Bundles Methods and System Designs” and the first version of the implementation was provided in the deliverable D2.2 “XMANAI Asset Management Bundles – First Release”. D2.3 as all previous deliverables in the work package 2 is a result of the coordinated work of in all its tasks. The deliverable provides detailed technical information for each of the components of the XMANAI Asset Management Bundles, including:

- The technical documentation of the provided APIs for each component.
- The technology stack and the implementation tools and libraries that were employed for the implementation of the first release, providing the details for the tools and libraries that were leveraged.
- The licensing information for the produced software artefacts and the access details for each AI Bundle.
- The main screenshots of the graphical user interface.
- A short summary of the improvements since the previous version.

The main improvements provided in the second release of the XMANAI Asset Management Bundles and presented in this deliverable are the following:

- The components File Data Harvester and Provenance Engine, which were previously only planned and designed are now implemented and included in the second release of the bundles.
- A graphical user interface was implemented for the components API Data Harvester and Policy Editor.
- The version control feature, where different versions of an asset can be stored and CRUD operations can be performed on these versions, was added to the Assets Store. The Asset Store developed for the previous release didn’t have this functionality., The support of several additional types of assets was added and the list of possible queries was extended.
- The APIs and the underlying methods of most components were further extended to enable their integration with other XMANAI platform components and the implementation of the planned workflows.

The next step is the integration of the XMANAI Asset Management Bundles developed in WP2 with the other services implemented in WP3 and WP4 under the second release of the XMANAI platform. It will be reported in the deliverable D5.3 “XMANAI Platform - Beta Version” on M32. After the end of WP2 the necessary improvements of XMANAI Asset Management Bundles addressing the new requirements that might occur in the future, through the feedback provided by the demo partners during the activities of WP6, will be done during the integration activities in WP5.

# References

XMANAI Deliverable D1.2 “XMANAI Concept Detailing, Initial Requirements, Usage Scenarios and Draft MVP”, 2021.

XMANAI Deliverable D1.3 “Updated Requirements and AI/Graph Analytics focused MVP”, 2022.

XMANAI Deliverable D1.4 “Final XMANAI MVP”, 2023.

XMANAI Deliverable D2.1 “Asset Management Bundles Methods and System Designs”, 2022

XMANAI Deliverable D2.2 “XMANAI Asset Management Bundles – First Release”, 2022

XMANAI Deliverable D3.1 “AI Bundles Methods and System Designs”, 2022

XMANAI Deliverable D3.2 “XMANAI AI Bundles – First Release”, 2022

XMANAI Deliverable D3.3 “XMANAI AI Bundles – Final Release”, 2023

XMANAI Deliverable D5.1 “System Architecture, Bundles Placement Plan and APIs Design”,2021.

XMANAI Deliverable D5.2 “XMANAI Platform- Alpha Version”, 2022.

# List of Acronyms/Abbreviations

| Acronym/ Abbreviation | Description |
|---|---|
| AI | Artificial Intelligence |
| ACID | Atomicity, Consistency, Isolation, and Durability |
| API | Application Programming Interface |
| AWS | Amazon Web Services |
| CRUD | Create, Read, Update, and Delete |
| CV | Cross Validation |
| DAG | Directed Acyclic Graph |
| DL | Deep Learning |
| DPE | Data Preparation Engine |
| EOE | Execution & Orchestration Engine |
| ETE | Experiment Tracking Engine |
| GPL | General Public License |
| GUI | Graphical User Interface |
| ID | Identity |
| IEET | Interactive Data Exploration & Experimentation Tool |
| ISO | International Organization for Standardization |
| JPEG | Joint Photographic Experts Group |
| JS | JavaScript |
| JSON | JavaScript Object Notation |
| KGM | Knowledge Graph Manager |
| MG | Model Guard |
| ML | Machine Learning |
| MSE | Mean Squared Error |
| PAAS | Platform As A Service |
| PD | Pipeline Designer |
| PDF | Portable Document Format |
| PSME | Pipeline Serving & Monitoring Engine |
| RDBMS | Relational Database Management System |
| Rest | Representational state transfer |

| | |
|---|---|
| RVE | Results Visualisation Engine |
| UI | User Interface |
| URL | Uniform Resource Locator |
| UUID | universally unique identifier |
| WP | Work Package |
| XAI | Explainable Artificial Intelligence |
| XMEE | XAI Model Engineering Engine |
| XMXE | XAI Models Explanations Engine |