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

D2.2: XMANAI Asset Management Bundles – First Release

Deliverable Leader: FRAUNHOFER

Due Date: M18

Dissemination Level: Public

Version: 1.0

Short Abstract

The current deliverable provides a detailed technical documentation for the first 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 documented information includes the implementation status of the first release, as well as the pending functionalities planned for the next release. Documentation of the API, the architecture and the technology stack that was employed for the development of the components are also reported, as well as 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 and the business users that can be informed of the provided functionalities and how to utilize them.

Further Information: www.ai4manufacturing.eu

Document Log

Contributors	**FRAUNHOFER**, ATHENA, POLIMI, SUITE5, TXT, TYRIS, UBITECH
Internal Reviewer 1	AIDEAS
Internal Reviewer 2	FORD
Type	Other

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 (AIDEAS)
R0.2	Revision of internal reviewer 2 (FORD)
R0.3	Revision of internal reviewer 3 (SUITE5)
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 first release of the XMANAI Asset Management Bundles components 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 deliverable provides information along the following dimensions: a) The implementation status of the first release of the component, according to the design specifications defined in deliverable D2.1 and any deviations from the specifications; b) The technical documentation of the provided APIs for each component; c) The technical details of the internal design and architecture of each component, as well as the interaction of each component with the other components of the overall architecture; 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) Screenshots of the core functionalities provided through the component's user interface where applicable; g) The implementation plan and pending functionalities to be delivered in the next final release of each component.

The first release of the WP2 "Industrial Asset Management and Secure Asset Sharing Bundles", which is documented in this report, will be integrated with the WP3 services under the first release of the XMANAI platform that will be reported in deliverable D5.2 "XMANAI Platform – Alpha Version" on M21. The development of XMANAI Industrial Asset Management and Secure Asset Sharing Bundles will continue, in order to provide additional features, based on the implementation plan and introduce required modifications and improvements of the existing component functionalities, that will occur in the future based on the feedback provided by the demo partners during the WP6 experimentation and testing activities. An additional release of the WP2 "XMANAI Industrial Asset Management and Secure Asset Sharing Bundles" will follow on M30 that will contain the pending features and improvements of the existing ones and will be documented in D2.3 "XMANAI Asset Management Bundles – Second Release".

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 any decisions or predictions due to the so-called “black-box effect”. Many of the machine learning/deep learning algorithms are opaque and not possible to be examined after their execution to understand how and 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 shall focus its research activities on Explainable AI (XAI) in order to make the AI models, step-by-step 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 D2.2 “XMANAI Asset Management Bundles – First Release” presents the first release of the XMANAI asset management components developed within all WP2 tasks on the basis of the design specification provided in the previous deliverable of this work package D2.1 “Asset Management Bundles Methods and System Designs”. The document provides a supportive documentation for the first release of the components that constitute the XMANAI Asset

Management Bundles: Assets Store, Provenance Information Store, 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 plan for the development of the additional functionalities. 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 next WP2 deliverable D2.3 "XMANAI Asset Management Bundles – Second Release", due in M30, will report on the final release of the Asset Management bundles.

1.3 Impact and Target Audiences

As in the case of D2.1 "Asset Management Bundles Methods and System Designs", 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 status of all the available functionalities of the XMANAI Asset Management bundles, the technology used, the provided interfaces as well as future implementations. 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, 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 first release.

1.5 Dependencies in XMANAI and Supporting Documents

This document provides a supportive report for the first 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". More specifically, D2.1 presented a design of the XMANAI Asset Management Bundles and the processes that will be supported, after a thorough state of the art review of the available technologies and approaches, in close collaboration with WP3 and WP5.

This deliverable will be followed by an additional release documented in D2.3 "XMANAI Asset Management Bundles – Second Release", which is due on M30 and will provide 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 functionalities for the overall XMANAI platform. Its original architecture was presented in the Deliverable D2.1.

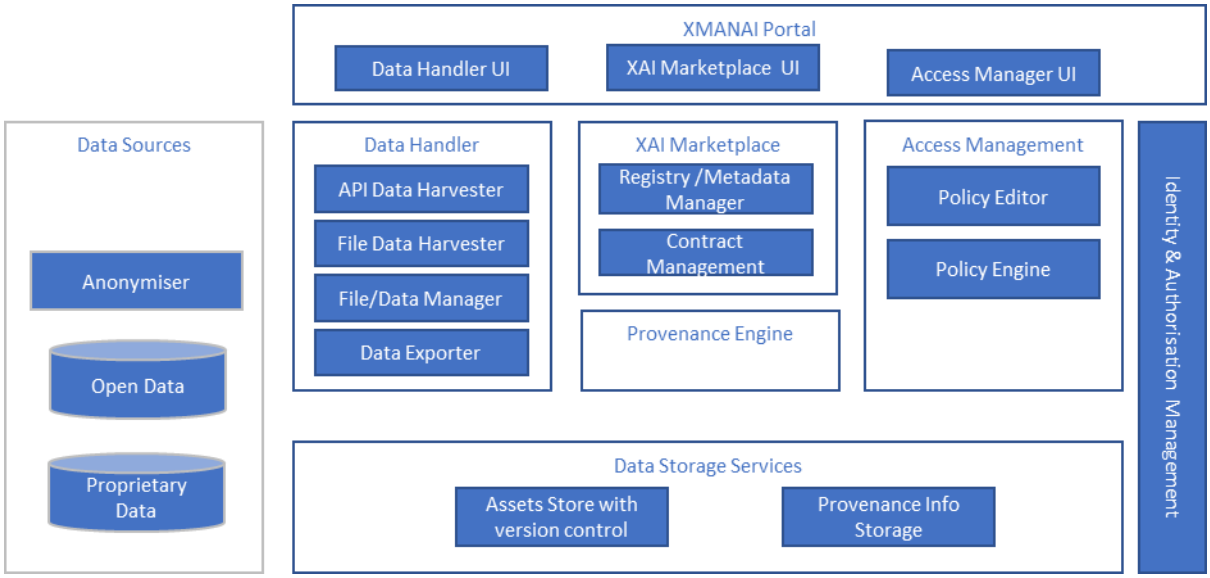

Figure 2-1: Overview of components providing industrial assets management and sharing functionalities

The assets are stored and managed by the Platform Storage services: Assets Store responsible for XMANAI assets and Provenance Information Storage for provenance information. The Data Handler enables data collection from external data sources to the Platform Storage Services, data export from XMANAI and access to the XMANAI data for external 3rd party services through an API. The Data Handler consists of:

- API Data Harvester collecting data from external APIs.
- File Data Harvester transforming data files to the XMANAI data model and saving data as XMANAI datasets in the storage.
- Data Exporter exporting data snapshots in requested file formats.
- File/Data Manager for managing data, models and other files associated with data analytics projects in XMANAI.

The Data Gateway sub-component of Data Handler planned in D2.1 to provide an API to get and push data from/to XMANAI data store is removed from the architecture to avoid any duplication of the functionalities since Asset Store can provide the required API.

The monitoring and logging of modifications and access to data will be done by the Provenance Engine. The Data Anonymizer is realised as a standalone tool executed on the data provider side. The data will be anonymised before sharing it with the XMANAI platform. The security mechanisms in the platform are realised with help of Identity and Access Manager, Policy Engine and Policy Editor. The data sharing between the data provider and data consumers are 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 chapters present the implementation status of the 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 component responsible for storing XMANAI assets. Version control as a feature will help to avoid losing data or other important information in the experimental multi-user data analytics XMANAI environment and if needed will help the user to get the required version of data or other XMANAI assets. The asset types stored in this component are:

- Structured Data conforming to XMANAI data model. Tabular data along with the data scheme are added to the Tabular Asset Store and where exists an API with endpoints to create, update and delete the tables.
- Files of different type including data files, data analysis or processing script files and any other files. The File Asset Store is responsible for storing binary files and it has an API, which provides endpoints for CRUD operations.

Binary and non-binary files are stored separately, and two separate databases are used for this purpose. Binary and non-binary files are stored as tabular data in two separate PostgreSQL databases which are connected to a REST API with endpoints for CRUD operations. Separate endpoints are provided for both type of files and a user can access individual endpoints to add and get files from the Assets Store.

Some important features of the Assets store are:

- Each of the files in the Assets Store is stored under a UUID (Universal Unique Identifier) which is generated by the database automatically while a file is created. This UUID is provided back to the user as a response to this operation. The name of the file and its corresponding UUID will be stored in a separate table or as part of the metadata. This is used to identify the table name with the UUID of a table.
- As mentioned earlier, new assets can be created. The created assets can be updated as well as deleted in the Assets Store. In addition to this, tabular assets can be previewed, and binary files can be downloaded from the Assets Store.
- Tabular data stored in the Assets Store also allows extraction of whole tables as well as entries of the table.
- Version control feature of the Assets Store allows to store all versions of a file if they are updated, and these versions can also be requested from the Assets Store. A version id is attached to each of the assets in this case to identify the individual versions.

Table 2-1: Assets Store – Implemented Functionalities

Feature ID	Functionalities & Implementation Status
AS.1	Assets Store for Binary Files This functionality allows the user and other XMANAI components to use the Assets Store to create and update binary files in the Assets Store. These files may include data files or script files. In this first release, a separate PostgreSQL database is attached to the Assets Store API for binary files. Each file is uploaded along with a scheme name, which creates a table for such type of files. More files can be added to the same tables or newer ones can be created with different scheme names. Each type of files, for example documents and images, will be stored in separate tables in the database for files. Each file is stored in separate rows in one table and each row has a UUID attached to it. Each file can thus be identified with a UUID.
AS.2	Assets Store for Non-Binary Files

Feature ID	Functionalities & Implementation Status
	This functionality allows the user and other XMANAI components to use the Assets Store to create and update non- binary files in the Assets Store. These files include structured data in the form of tables with a specific data scheme for column names. In this first release, a PostgreSQL database with a flask API is implemented with endpoints to add, update, retrieve and delete whole tables as well as individual entries of the tables. Each file is saved in the database under a UUID which replaces the original table names added by the user. After each table is created, the UUID of the table is sent back to the user as a response. This UUID and its corresponding table names will be stored in a separate table or in the metadata store, this functionality will be implemented in the later release.

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
Binary Files		
/assets/files	POST GET DELETE	Add a new file to the Asset Store. Get an asset from the Asset Store by asset UUID and scheme name. Delete an asset from the Asset Store by asset UUID and scheme name.
/assets/files/{asset_uuid}/{version_id}	GET	Get a specific version of an asset from the Asset Store by asset UUID, scheme name and version id.
Non- Binary Files		
/asset	POST	Add a new asset or update an existing asset in the Asset Store.
/assets	POST GET DELETE	View several asset tables present in the Asset Store. Get an asset table by asset UUID or entries from an asset table by asset UUID and entry id. Delete several assets by asset UUIDs from the Asset Store.

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

Endpoint	Method	Description
/assets/{asset_uuid}	PUT DELETE	Update some entries of an asset table by asset UUID Deletes several entries from a table using the asset UUID and entry ids.
/asset/{asset_uuid}/{version_id}	GET	Get a specific version of an asset by the asset UUID and version id.

2.2.3 User Interface

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

2.2.4 Technology stack and License information

The Assets Store has a PostgreSQL database with a REST API attached to it to perform all data manipulation operations in the Store. The API is written in Python with the Flask framework. All components can be published under Apache-2.0-license.

2.2.5 Code repository

The code of the component is published in a private section of the XMANAI repository in GitLab[2], access to which can be provided by a request sent to the project coordinator.

2.2.6 Pending functionalities roadmap

The table below presents the planned functionalities planned for the next release of the component.

Table 2-3: Assets Store – Pending Functionalities

Feature ID	Pending Functionalities
AS.1	Assets Store for Binary Files The database used for Binary Files is also a PostgreSQL database. As the functionalities of the asset store improve, if larger files need to be added to it, a suitable database will be selected.
AS.2	Assets Store for Non-Binary Files Similar to the File Assets Store, the UUID generated by the Tabular Assets Store is stored along with the original table names in a database. This feature will be simultaneously implemented for both the Assets Stores. Along with this, the REST API of the Assets Store will be extended to add an endpoint to provide a list of all table names to the File Data Harvester. This will be continued to facilitate further communication with other components.
AS.3	Version Control for Assets Store

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

Feature ID	Pending Functionalities
	Version control feature for the Assets Store is used to identify and store separate versions of a file if any modification was performed by the user. The version control feature is supported by the use of a version id, which is attached to every file present in the Assets Store. The version id is updated every time a file is updated by the user. For the tabular data, the level of versioning can be configured by the user which lets them decide if every small change made to the table needs to be stored as a new version of a file or if only a complete update to the table requires a new version id. In this first release, the version control feature is not implemented in the File Assets Store as well as the Tabular Assets Store.

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.

2.3 Data Storage Services: Provenance Information Store

2.3.1 Description

Provenance Information Store is the core component responsible for storing the provenance information of an asset stored in the Assets Store (ref). Each CRUD operation performed on the asset will result in a provenance entry which describes the Agent performing the operation, the Entity the operation is performed on and the Activity which is being performed. The metadata schema developed by the World Wide Web Consortium (W3C) is used for this purpose. A detailed description of this was provided in Deliverable 2.1, Section 2.2.12.

The Provenance Information Store is a database consisting of a REST API for adding and retrieving provenance information. Each entry is added to the Assets Store. All provenance information can be retrieved from the Provenance Information Store through the GET request of the API. Since W3C PROV uses the Resource Description Framework format (RDF), the database in Provenance Information Store is a triplestore. The database solution chosen for this is Apache Jena Fuseki, which also contains a SPARQL endpoint to query provenance information from the store. The API of the provenance information store thus collects requests and converts them into SPQRQL queries to access the triplestore.

The provenance information being added to the Provenance Information Store is created by the Provenance Engine which is described in Section 2.10. The output of this component is a provenance document in the form of a JSON file that describes all entities and attributes described defined in the W3C PROV model. The Provenance Information Store collects this data and converts it to the SPARQL query for the triplestore to add a provenance entry. An example SPARQL query is provided below.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT DATA
{
  <http://example/book1> dc:title "Asset Name";
                         dc:creator "Asset Publisher".
}
```

At the moment of writing, only the API documentation of this component is available and its development has not started. The initial code release of this component will be in Deliverable D2.3. It would include all the above-mentioned functionalities of the Provenance Information Store.

2.3.2 API documentation

The Provenance Information Store has a REST API attached to it that assists in creating and retrieving provenance information. The swagger file for the API documentation is present in the project's GitLab repository[3] and a brief description of it is provided in the following table.

Table 2-4: Provenance Information Store – API Endpoints Overview

Endpoint	Method	Description
/provenance/{asset_uuid}	POST GET	Create a provenance entry for an asset Get provenance information of an asset
/provenance/{asset_uuid}/{entry_id}	GET	Get one provenance entry of an asset

2.3.3 User Interface

This component does not have a graphical user interface (GUI).

2.3.4 Technology stack and License information

The triplestore solution used for this component is Apache Jena Fuseki. A REST API written in Python with Flask framework will be attached to it for adding and retrieving provenance information. All components can be published under Apache-2.0-license.

2.3.5 Code repository

This information will be released once the development of the component has progressed.

2.3.6 Pending functionalities roadmap

The table below presents the planned functionalities planned for the next release of the component.

Table 2-5: Provenance Information Store – Pending Functionalities

Feature ID	Functionalities & Implementation Status
PIS.1	Create provenance entry This functionality allows the Assets Store to send a POST request to the Provenance Information Store every time a new asset is created or any modification has been performed on an existing asset in the Assets Store. As mentioned earlier, the provenance document is created by the Provenance Engine and the API of Provenance Information Store converts this information to a SPARQL query that

[3] https://GitLab.com/xmanai-h2020/api-docs/-/blob/main/Provenance Information Storage.yaml

Feature ID	Functionalities & Implementation Status
	inserts the data into the triplestore. In this first release, this functionality is not fully developed.
PIS.2	Get provenance entry This functionality allows any of the XMANAI components to get provenance information of an asset. A component can access the GET request of the API of the Provenance Information Store to retrieve all the provenance information of that particular asset. This feature will be further extended based on the provenance requirements of other XMANAI components.

2.3.7 Considerations

At the time of writing, no foreseen challenges or limitations are present for proceeding with the development of the Provenance Information Store.

2.4 Data Handler: API Data Harvester

2.4.1 Description

The API Data Harvester serves the purpose to harvest data from data providing APIs, transforming it to the XMANAI data standard and ingesting it into the Asset Store. A harvesting-process can either be triggered immediately and manually or at predefined time intervals in an automated manner in order to allow for a periodic updating of the data. It consists of five components, which communicate among themselves via HTTPS 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 specified in the endpoint URI. The **Pipe Object** is a JSON-file, which contains all relevant information in regards to the harvesting process, such as the API endpoint of the data source and the name of the asset to be created. 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**. All **Pipe Objects** are to be stored in a Git repository, which is then to be specified as an environment variable for the **Scheduler**. When being triggered, it selects the specified **Pipe Object** and sends it to the **Importer**.
- The **Importer** receives the **Pipe Object**, queries the endpoint mentioned 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 and 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, which is being received as a response of the Asset Store is subsequently being stored in the **UUID Store**.

- In order to keep track of the assets and their corresponding **Pipe Objects** the respective UUID of newly created assets are being stored in the **UUID Store** together with the name of the corresponding **Pipe Object**. This enables an updating of already existing assets instead of creating new ones every time a new harvesting process is being triggered.

Table 2-6: API Data Harvester – Implemented Functionalities

Feature ID	Functionalities & Implementation Status
ADH.1	Creation and Management of **Pipe Objects** (implemented): • Creating Pipe Objects and managing them in a Git repository
ADH.2	Use case specific Configuration of the Data Harvesting Parameters through **Pipe Objects** (implemented): • Selecting individual API endpoints for data harvesting • Configuring individual data-transformation/-cleansing/-anonymization • Configuring metadata and the data model
ADH.3	Time specific execution of Harvesting Processes through **Triggers** (implemented): • Either manual and immediate or automated and interwall-wise triggering of Harvesting Processes • Creating a new asset • Updating an existing asset

2.4.2 API documentation

For the **Scheduler** the API documentation is to be found in the projects GitLab repository[4]. The table below provides a brief overview as well:

Table 2-7 API Data Harvester – API Endpoints Overview

Endpoint	Method	Description
/triggers	GET	Get a list of all **Triggers**.
	PUT	Bulk update all **Triggers**.
/triggers/{pipeID}	GET	Returns all **Triggers** of the specified **Pipe Object**.
	PUT	Create or update **Trigger** for the specified **Pipe Object**.
	DELETE	Delete previously created **Triggers** for the specified **Pipe Object**.
/triggers/{pipeID}/{triggerID}/{status}	GET	Set status of specified **Trigger** for specified **Pipe Object**.

[4] 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[5]. The table below provides a short overview regarding the most important attributes of this data object:

Table 2-8: API Data Harvester – Pipe Object Attributes

Component	Attribute	Description
Importer	body/config/address	API-endpoint to harvest the data from
	body/config/action	Action to be performed. Currently available: • create • update
	body/config/pipeName	Name of the Pipe Object associated with the Harvesting Process also referred to as the **pipeID** when addressing the **Scheduler**.
Transformer	body/config/scriptType	Possible values: • embedded • repository
	body/config/repository	Only needed if "scriptType "is set to "repository". Describes location of repository and transformation script.
	body/config/repository/script	Describes path to the transformation script within the repository.
	body/config/script	Only needed if "scriptType" is set to "embedded". Contans JS-code of the transformation script itself.

2.4.3 User Interface

The component doesn't provide a GUI in this version.

2.4.4 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

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

event-based manner, utilizing the Vert.x-Library[6] and allows several thousand requests to be handled simultaneously. The UUID Store is a simple PostgreSQL database. All components can be published under Apache-2.0-license.

2.4.5 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.

2.4.6 Pending functionalities roadmap

The table below presents the planned functionalities planned for the next release of the component.

Table 2-9: API Data Harvester – Pending Functionalities

Feature ID	Functionalities & Implementation Status
ADH.4	Offering an additional endpoint to be addressed by the Asset Store every time an asset, which was previously created by the **Api Data Harvester**, is being deleted through another component. Only this way the **UUID Store** can stay updated.

2.4.7 Considerations

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

2.5 Data Handler: File Data Harvester

2.5.1 Description

The File Data Harvester is a component responsible for transforming data from file(s) to a structured form according to the XMANAI data model and submitting it for storing to the XMANAI Assets Store. It has a user interface which helps the user to describe the semantics and structure of a data file. The user can select a file which is already present in the XMANAI File Asset Store and add information to it such as the dataset name and description along with some keywords. It further allows the user to provide a metadata of the file. The final output of this component is a dataset that will be added to in the Assets Store.

At the moment of writing, only an API documentation of this component is available and the its development of it has not started. The initial code release of this component will be in Deliverable D2.3.

2.5.2 API documentation

A brief overview of the endpoints of the File Data Harvester is present in the following table and the swagger file for the API documentation is present in the project code repository[8].

[6] https://github.com/vert-x3/vertx-lang-kotlin
[7] https://GitLab.com/xmanai-h2020
[8] https://gitlab.com/xmanai-h2020/api-docs/-/blob/main/File data harvester.yaml

Table 2-10: File Data Harvester – API Endpoints Overview

Endpoint	Method	Description
/datasets	GET	Get all existing datasets from the database for the user to choose
/datasets/keyword	GET	Get all existing keywords for the dataset
/datasets/category	GET	Get all existing categories for the dataset
/datasets/publisher	GET	Get all existing publishers for the dataset
/datasets/project	GET	Get all existing projects for the dataset
/datamodel/{dataset}	POST	Add a dataset datamodel

2.5.3 User Interface

The mockups of the GUI interface of this component were presented in the previous deliverable 2.1. For this initial release, these mockups have not been developed.

2.5.4 Technology stack and License information

This information will be released once the development of the component has progressed.

2.5.5 Code repository

This information will be released once the development of the component has progressed.

2.5.6 Pending functionalities roadmap

The table below presents the planned functionalities planned for the next release of the component.

Table 2-11: File Data Harvester – Pending Functionalities

Feature ID	Functionalities & Implementation Status
FDH.1	Preview of the data This component will allow the user to view the dataset initially so that its structure and semantics can be understood for further allocation of metadata. In this release, this feature is not implemented as part of the File data harvester.
FDH:2	Ability to describe the type of the data using the XMANAI data model as a reference. For tabular data, each of the columns will be allocated with a datatype by the user from the XMANAI data model. In this release, this feature is not implemented as part of the File data harvester.

Feature ID	Functionalities & Implementation Status
FDH.3	Ability to provide the basic metadata (name, description, categories, etc.) for the dataset This functionality will be developed in alignment with the Registry/Metadata Manager, which is not developed for this release.
FDH.4	Save the provided metadata in the metadata registry and create a suitable data structure to store data in the datastore 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	Ability to add data from a file(s) to an existing dataset This functionality will help more data to be added to an existing dataset from a different data file. This feature is not developed for this release of the File Data Harvester.

2.5.7 Considerations

- The API of the File Data Harvester enables users to describe the structure and semantics of the data in the selected files and provide additional metadata to create out of these files well-defined and structured datasets, which could be queried through the Assets Store in XMANAI. For describing the data from files, the component will use the elements of the XMANAI data model. This brings some requirements to the XMANAI data model and to the API for searching and obtaining its elements.
- In Assets Store the assets are stored under an UUID. The assets' names and other metadata have to be stored in the Metadata Registry of the Marketplace. An API for doing that is required.

2.6 Data Handler: File/Data Manager

2.6.1 Description

The File/Data Manager (FDM) provides an overview of the assets belonging to a user. When the user logs into the system, they want to get an overview of their previous work. The File/Data Manager will show all the user's assets organized into projects. The user may select from a list of available actions from this interface, like choosing a dataset or using the Interactive Exploration Component for experimentation.

The File/Data Manager interacts with other components to provide the overview mentioned above. The component will interact with the Identity & Authorisation Management to identify the current user and check its credentials. Afterwards, FDM interacts with the Registry/Metadata Manager to get the list with all the relevant assets.

Although we cannot prescribe the list of available actions toat a full extent, the following capabilities should be included:

Table 2-12: File/Data Manager – Implemented Functionalities

Feature ID	Functionalities & Implementation Status
FDM.1	Overview of the datasets that are accessible to the current user.

	This functionality is fully supported in the current development stage of the component. As we will present later in the User Interface section, the user can obtain an overview of all their available assets, such as files, datasets, models etc.
FDM.2	Search in the list of the available datasets. The component will provide the functionality for the user to search among the list of his the available datasets. This functionality is fully supported in the current development stage of the component. The user can search and filter some of their available assets based on a keyword.
FDM.3	Select an available dataset and rename it. The user should select a dataset that he has access to and then rename it. 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 dataset. The user will be able to create a new dataset and then insert information that describes it. 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 one is about uploading a file from the user's local pc to the XMANAI application. FDM fully supports this action. The second one is about transforming a file into a dataset. This action is performed by the File Data Harvester. The FDM provides this functionality by calling the File Data Harvester.
FDM.5	Delete an available dataset This functionality is fully supported in the current development stage of the component. The user can select a component and delete it.

2.6.2 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.

2.6.3 User Interface

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

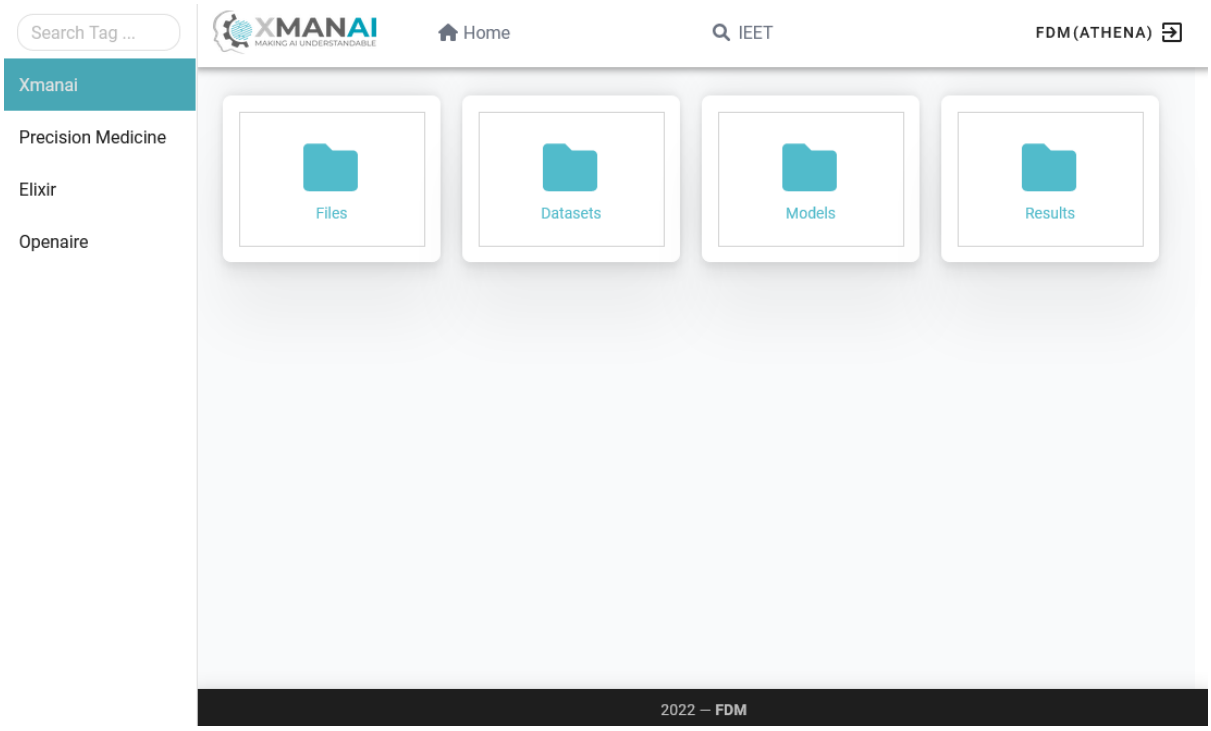

Figure 2-2: An example of the assets of a user. The assets are organized into Files, Datasets, Models and Results.

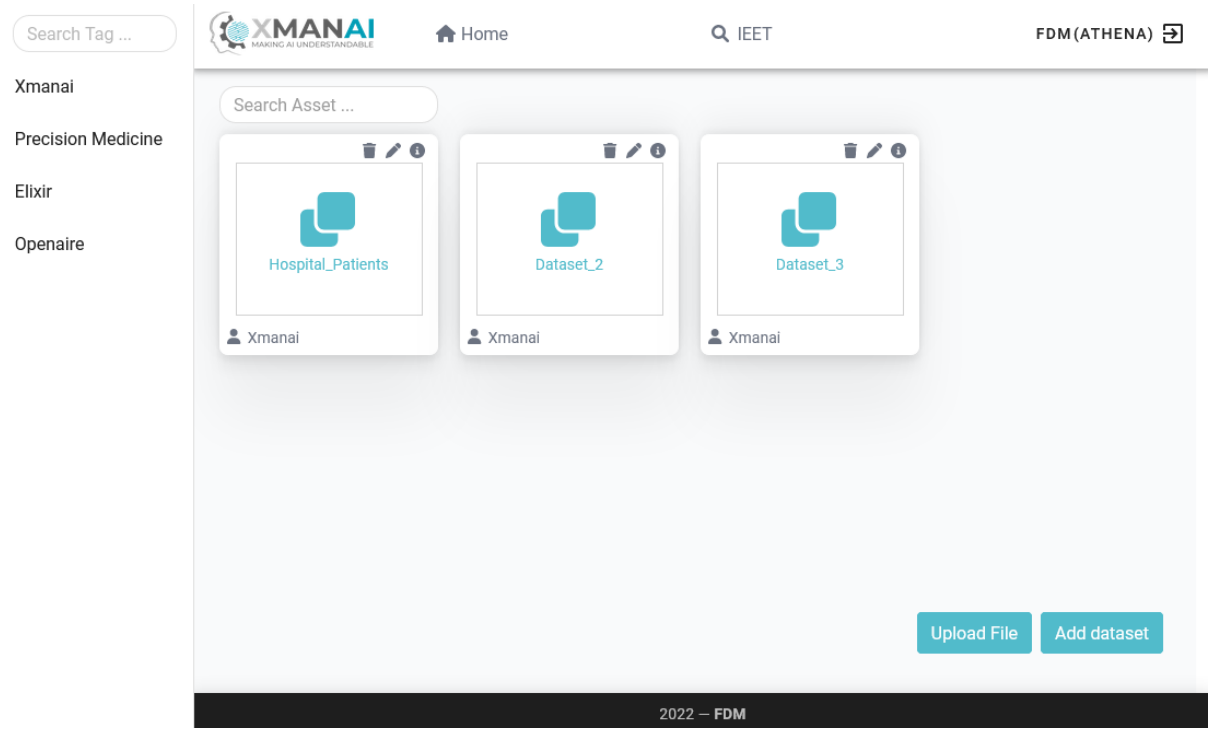

Figure 2-3: An example of the files of a user inside a project.

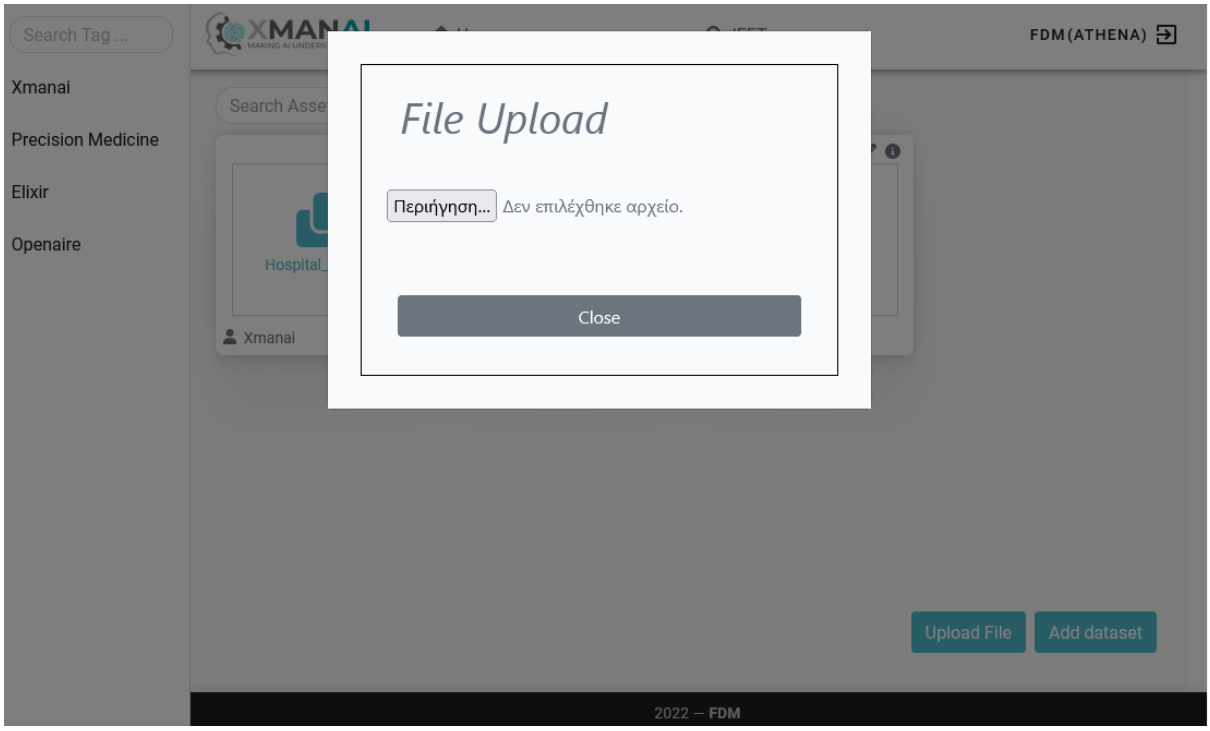

Figure 2-4: The GUI for uploading a file to XMANAI

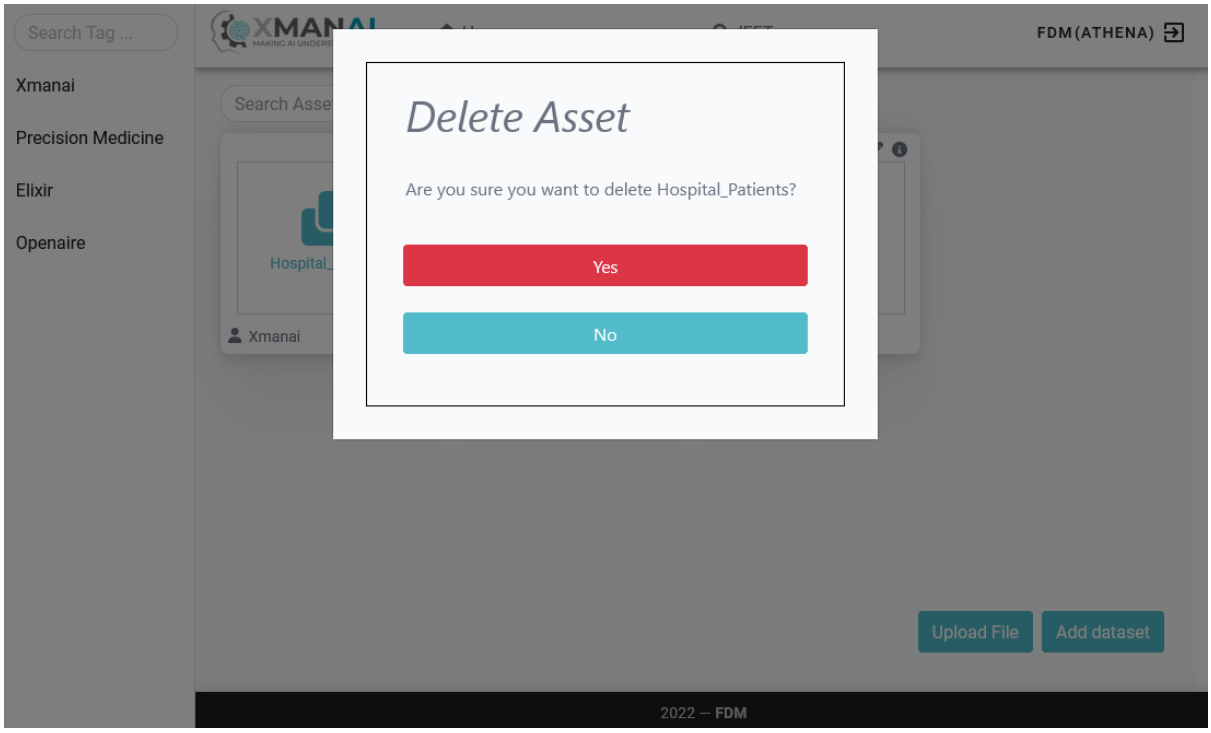

Figure 2-5: The interface for deleting an asset.

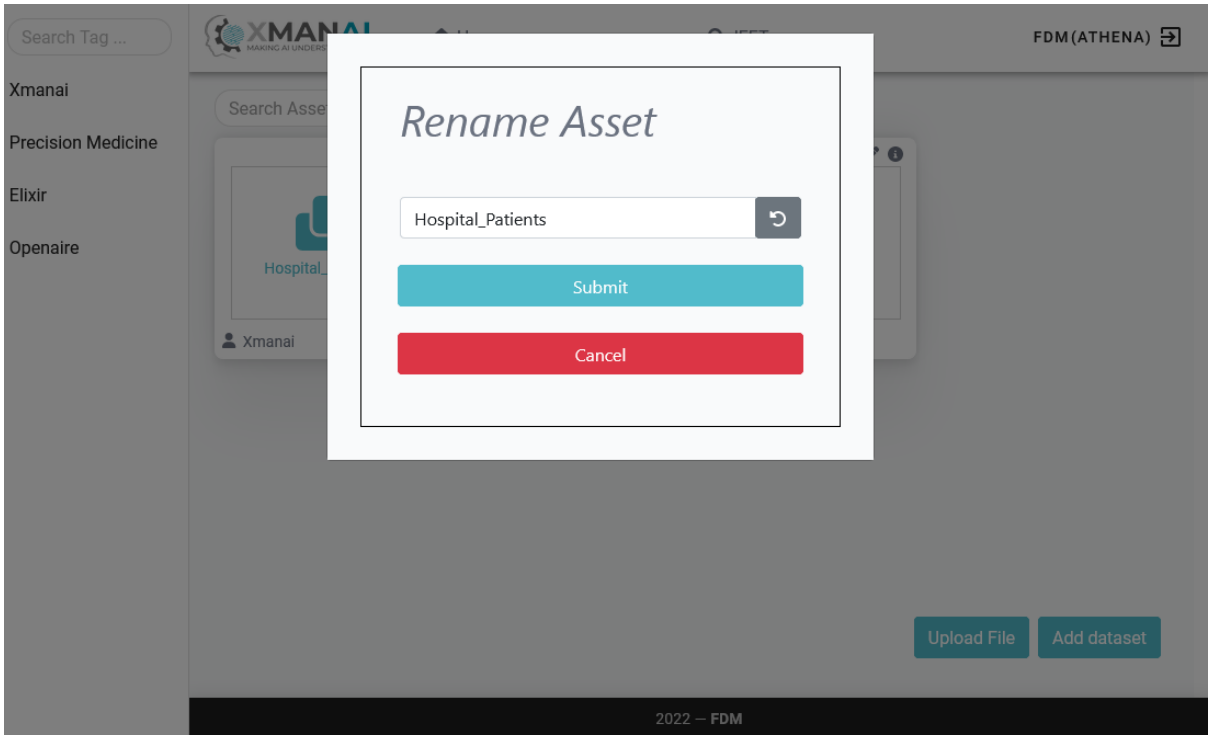

Figure 2-6: The interface for renaming an asset

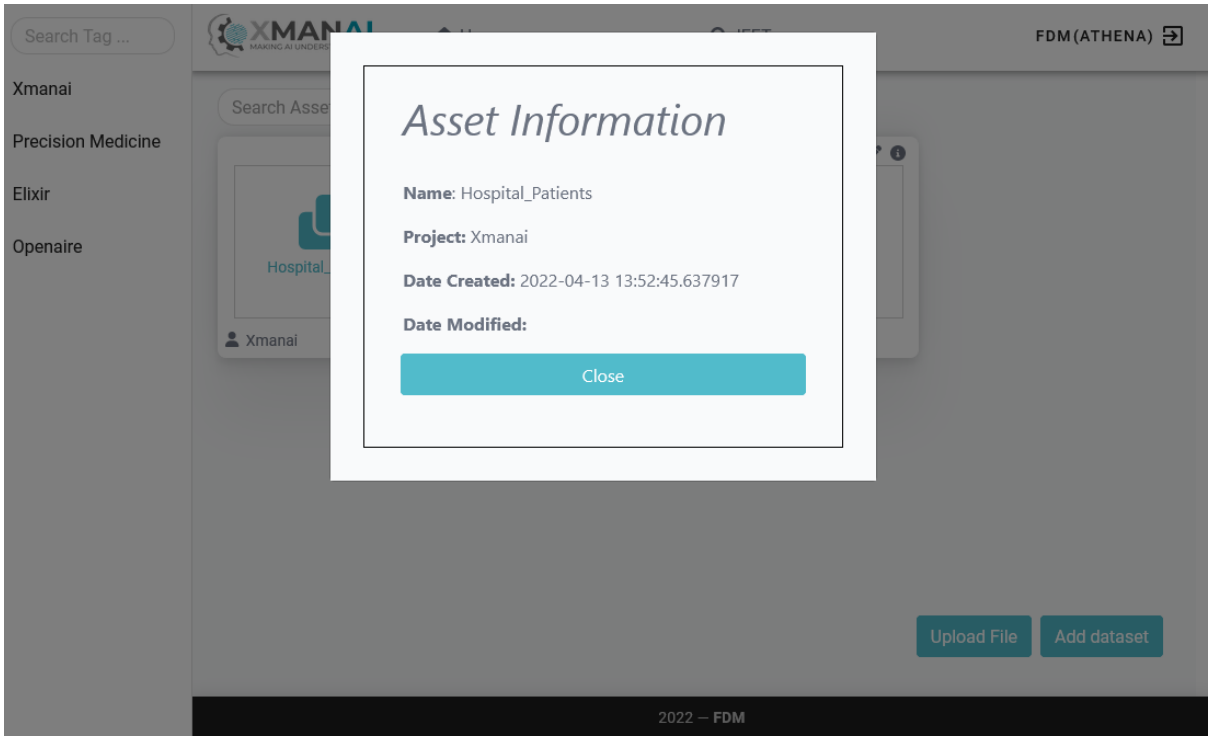

Figure 2-7: The interface for displaying information about an asset

2.6.4 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 component with Vue.js JavaScript library. The front-end and the back-end are developed inside Docker containers. This version of the component is proprietary.

2.6.5 Code repository

The code of the component is published in a private section of the XMANAI repository in GitLab[9], access to which can be provided by a request sent to the project coordinator.

2.6.6 Pending functionalities roadmap

As described in Section 2.6.1, the FDM covers all the appropriate functionalities in the current development circle. Therefore, for the time being, there are no pending functionalities. However, since FDM is related to other components, new requirements can arise during the next months.

2.6.7 Considerations

We describe some considerations regarding the FDM:

- The FDM functionalities seem to have some overlapping with the XAI Marketplace. For example, both components provide the user with an overview of its datasets. Therefore, during the platform integration activities, we shall clarify the overlapping functionalities to ensure a seamless and clear user experience.

2.7 Data Handler: Data Exporter

2.7.1 Description

The Data Exporter is the component responsible for all the data local saving functionalities. It mainly creates slices of the data that is gathered into the XMANAI platform and saves file instances of them in different accessible formats (e.g.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 2-13: 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: • Accessing to different datasets. • The list of available datasets as a table of available datasets.

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

Feature ID	Functionalities & Implementation Status
	• 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: The ability to select between different output formats. For this release, as the initial dataset will be in tabular format, only CSV format is provided. The capability to select a path to store the file into the local system. The generation of a file with the data of the dataset using the specified output format.

2.7.2 API documentation

The component doesn't does not provide an API.

2.7.3 User Interface

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

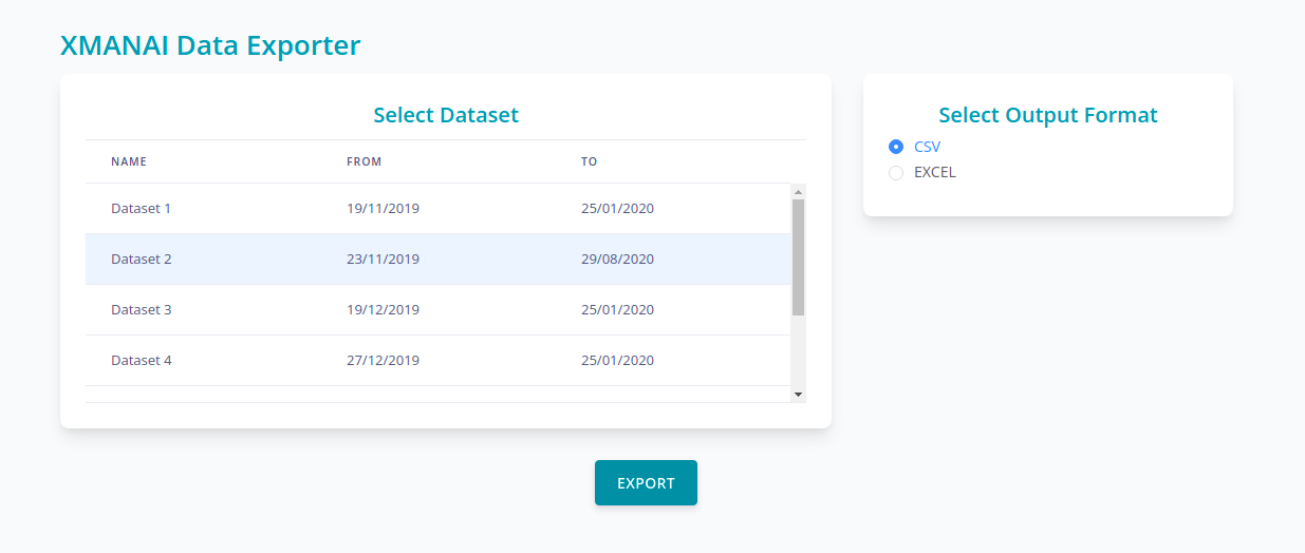

Figure 2-8: User interface of XMANAI Data Exporter

2.7.4 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 have been used.
- Nginx. It serves as a proxy and directs client requests to the appropriate component.

According to the information provided in D7.2, this component has a closed source license and an agreement with Tyris must be reached in order to use it.

2.7.5 Code repository

The code of the component is published in a private section of the XMANAI repository in GitLab[10], access to which can be provided by a request sent to the project coordinator.

2.7.6 Pending functionalities roadmap

The table below presents the planned functionalities planned for the next release of the component.

Table 2-14: Data Exporter – Pending 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.
DE.2	Generate dataset slices by selecting a specific data range (i.e. initial and end dates) and a concrete version of the data from the version control. The ability of filtering a dataset between a data range is excluded from this version and will be available in the next release.
DE.3	**Convert the queried data into a local file.** At this moment, only the generation to CSV format is provided. It is intended to support more formats. At least, the generation of files in JSON format to support non-tabular datasets must be provided in the next release.

2.7.7 Considerations

As no other component depends on the Data Exporter, the provision of endpoints for downloading datasets has been excluded. This is due to the fact that these datasets are expected to be downloaded always using the graphical interface.

2.8 XAI Marketplace: Registry/Metadata Manager

2.8.1 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, pipeline or 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:

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

Table 2-15: XAI Marketplace Metadata Schema

Metadata Name	Definition	Category	Related Standard(s)
Identifier	A unique identifier of the asset.	Dataset, Model	DCMI, DCAT
Title	A name given to the specific asset.	Dataset, Model	DCMI, DCAT
Description	A free-text account of the specific asset.	Dataset, Model	DCMI, DCAT
Type	The nature or genre of the asset (i.e. dataset, model, ...)	Dataset, Model	
Theme	A main category of the asset.	Dataset, Model	DCAT
Publisher	The entity responsible for making the item available.	Dataset, Model	DCMI, DCAT
Keywords	A keyword or tag describing the asset.	Dataset, Model	
Temporal Coverage	The temporal period that the dataset covers.	Dataset	DCMI, DCAT
Spatial Coverage	The geographical area covered by the dataset.	Dataset	DCMI, DCAT
Purpose	The problem for which the model was created to address.	Model	
Library	The ML/DL library in which the model was created.	Model	
Explainability Technique	The main explainability techniques leveraged in the specific model.	Model	
Date Created	The date when the asset was created/released.	Dataset, Model	DCMI, DCAT
Date Updated	Most recent date on which the asset was changed, updated or modified.	Dataset, Model	DCMI, DCAT
Access Rights	An indication of the asset's security status.	Dataset, Model	
License	A license, a legal document under which the asset is made available.	Dataset, Model	DCMI, DCAT
Version	The version of the specific asset.	Dataset, Model	DCMI

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, etc.) or be provided in collaboration with other XMANAI components, such as the Data Harvester or the XAI Pipeline Manager. It should be noted that the *access rights* feature defines whether the asset can be shared or not in the marketplace.

The following table summarizes the implemented features in this release.

Table 2-16: 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 and AI models and can be further analysed into four management subfunctions: • Initial definition/creation of metadata for a data asset • Enrichment or update of metadata for a data asset

Feature ID	Functionalities & Implementation Status
	• Retrieval of metadata for a data asset • Removal of metadata for a data 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 datasets and AI 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 and the Provenance 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: • Text search • Filtering • Sorting

2.8.2 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[11]:

Table 2-17: Registry/Metadata Manager – API Endpoints Overview

Endpoint	Method	Description
/api/v1/marketplace/data-assets	GET	Retrieve a list of the available data assets (dataset, file, model)
/api/v1/marketplace/data-assets/{id}	GET	Retrieve the metadata of a data asset (dataset, file, model) by its id.
/api/v1/marketplace/data-assets/{id}	PUT	Update the metadata of a data asset.
/api/v1/marketplace/data-assets/{id}	DELETE	Delete the metadata of a data asset by its id.

2.8.3 User Interface

As depicted in the following figure, the user visits the XAI Marketplace where he/she may view the available datasets and AI models 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.

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

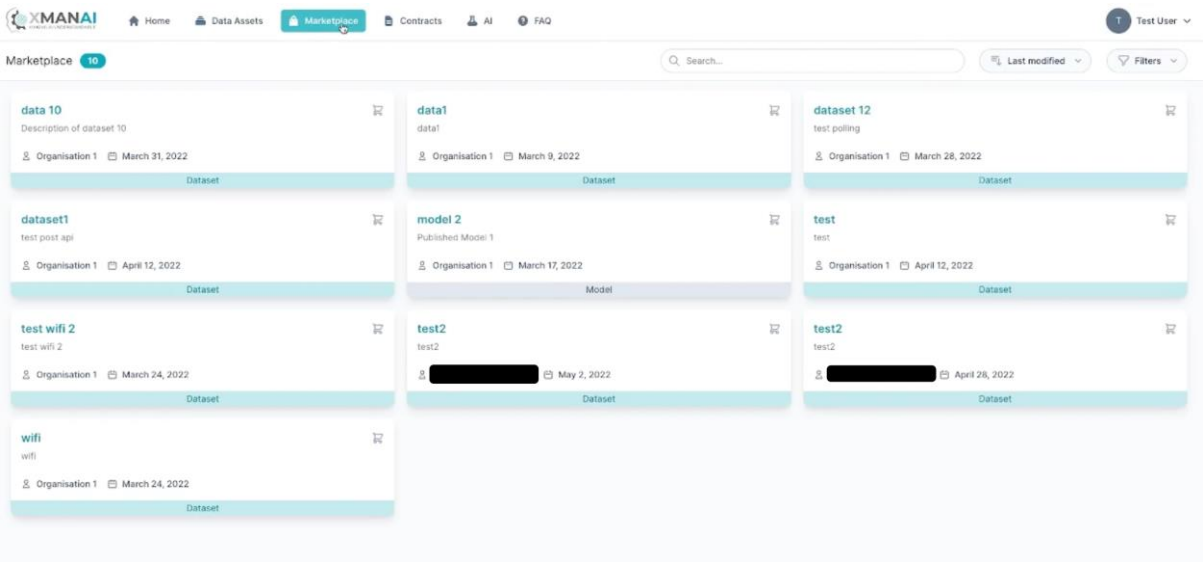

Figure 2-9: XAI Marketplace - Registry/Metadata Manager. Main Page

By selecting a specific asset, the user may view its detailed metadata as depicted in the following figure.

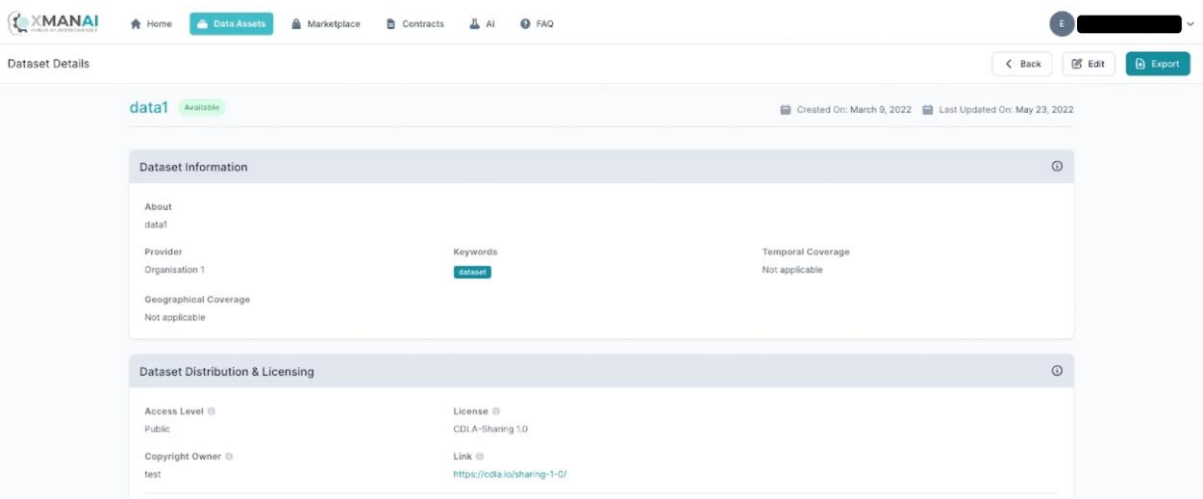

Figure 2-10: XAI Marketplace - Registry/Metadata Manager. View Asset Details

2.8.4 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 2-15: 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[12] JavaScript framework, released under the MIT License.

[12] https://vuejs.org/

2.8.5 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.

2.8.6 Pending functionalities roadmap

In the current release, only datasets and models are considered as assets to be handled by the Registry/Metadata Manager and be available in the catalogue. Results, experiments and XAI pipelines will be added during the development of the final version of this component. Additional metadata concerning datasets and models shall be added on demand depending on any additional needs/requirements by the users (XMANAI demonstrators or other XMANAI components).

2.8.7 Considerations

In this first release, the Registry/Metadata Manager comes as a fully functional component that handles datasets and AI models, the two most common types of assets. Additional asset types, as already mentioned, will be introduced in the final iteration since these two artefacts can be the outcome of multiple datasets and/or AI models, the combination of which, may introduce several challenges from a technical as well as from an IPR perspective.

However, the most important challenge for this component is the integration with the XMANAI platform and its interaction with other XAI components. For instance, the Registry/Metadata Manager receives parts of the metadata information (e.g., id, asset title, date created, etc) in collaboration with the Data Harvester and XAI Model Catalogue. The data assets presented to the user in the catalogue view are automatically filtered based on the respective access policies and IPRs, which are communicated by the Policy Engine and the Provenance Engine.

All these challenges have been foreseen during the initial development of the Registry/Metadata Manager so that the final version of the XMANAI platform will have them addressed in the most efficient manner. Moreover, the feedback that will be provided by the end-users will also shape the final steps of the development and refine all functionalities.

2.9 XAI Marketplace: Contract Manager

2.9.1 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 or be renewed, if all parties are in agreement. 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 2-18: 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 into 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 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.

2.9.2 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[13].

Table 2-19: 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 /{id}	GET	Retrieve a specific contract by its id.
/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.

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

Endpoint	Method	Description
/api/v1/marketplace/contracts/{id} /reject	PATCH	Reject the terms of a specific version of a contract.

2.9.3 User Interface

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.

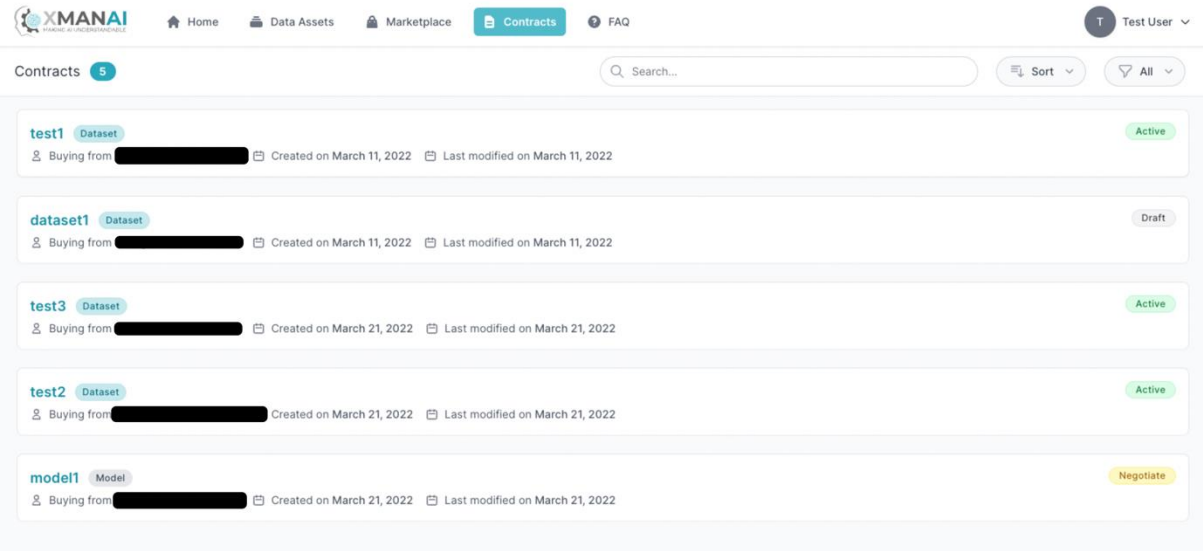

Figure 2-11: XAI Marketplace - Contract Manager. View Contracts List

When a request for an asset is created, the asset provider can prepare a contract, defining the terms and the asset permissions in terms of read, write and share access as depicted in the following figure. In order to sign the contract, the asset provider needs to unlock his/her wallet.

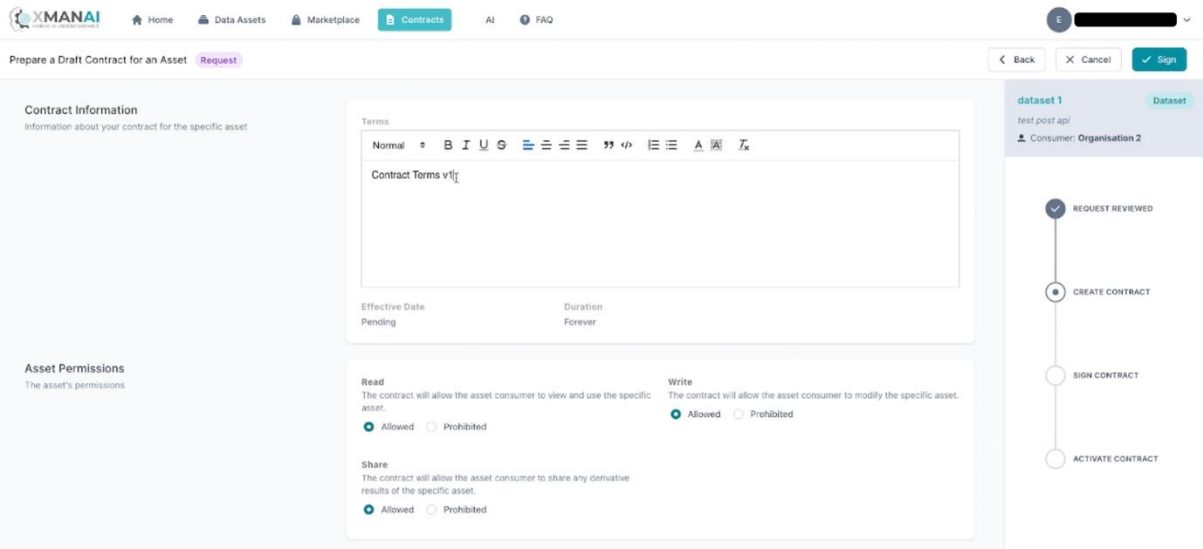

Figure 2-12: XAI Marketplace - Contract Manager. Create a new Contract

The asset consumer can view the draft contract and decide whether he/she accepts/rejects the terms or opts to negotiate. In the case of negotiation, the changes introduced by the asset consumer are immediately visible through a “track change” mechanism as depicted in the following figure.

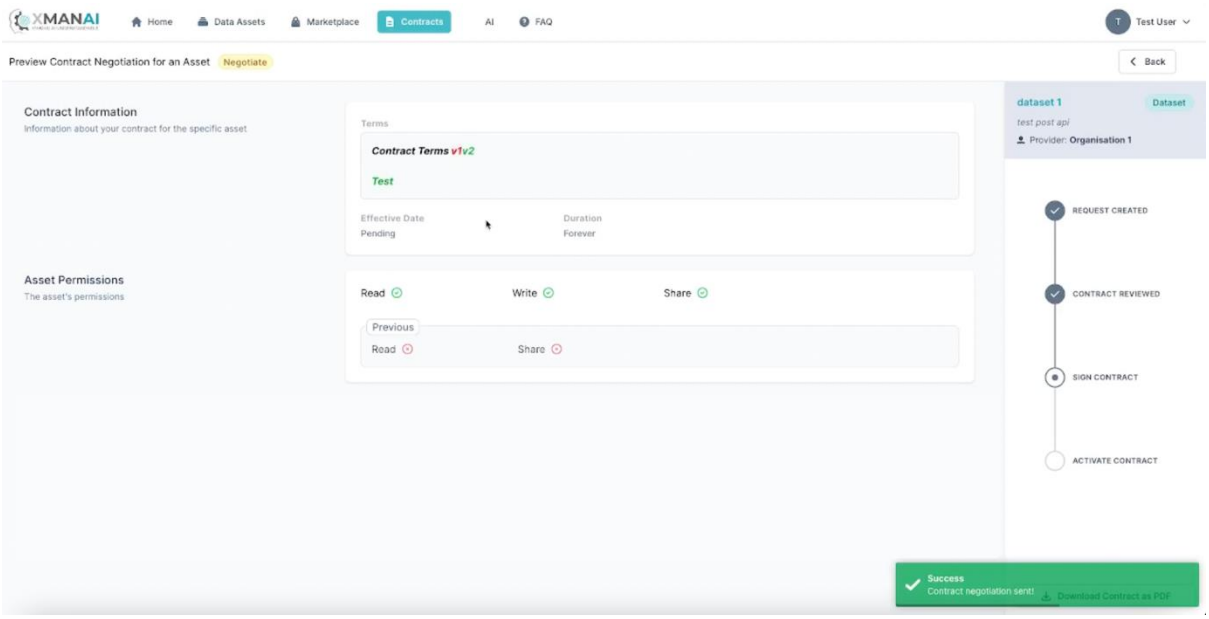

Figure 2-13: XAI Marketplace - Contract Manager. Negotiate a Contract

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.

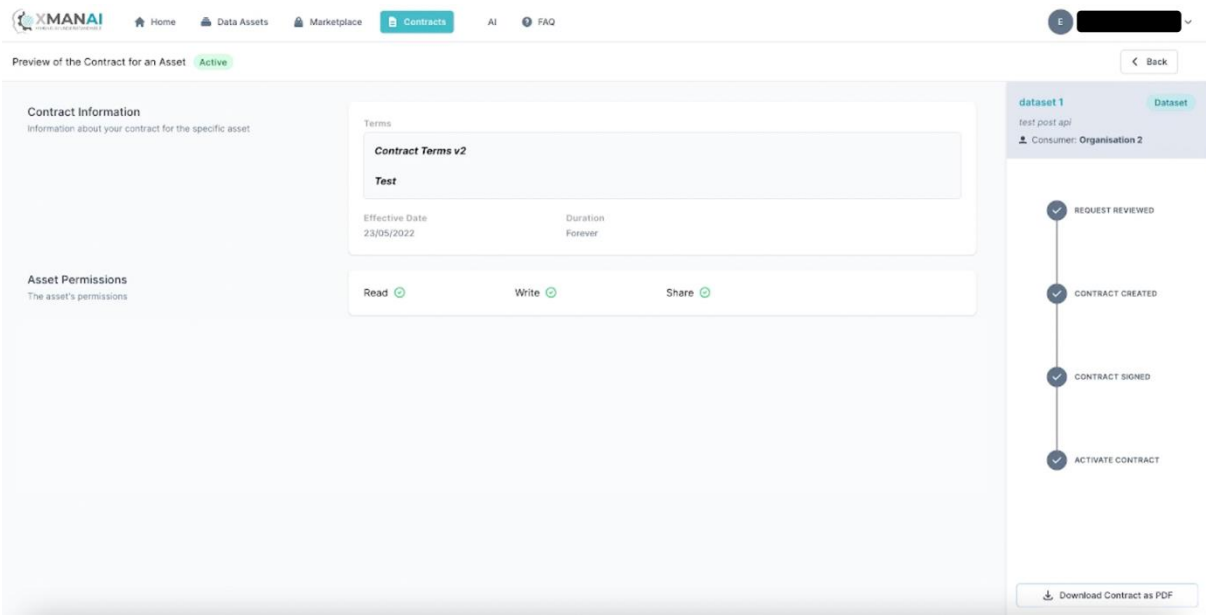

Figure 2-14: XAI Marketplace - Contract Manager. View Contract Details

2.9.4 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 VueJS 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.

2.9.5 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.

2.9.6 Pending functionalities roadmap

In the current release, only datasets and AI models are considered as assets to be available for inclusion in a smart contract. Results, experiments and pipelines will be added during the development of the final version due to the inheritance relations among the assets that need to be further investigated.

2.9.7 Considerations

The first release of the Contract Manager includes the complete set of functionalities for handling sharing agreements over datasets and AI models. As in the case of the Registry/Metadata Manager, additional asset types (e.g. results, experiments) will be introduced in the final iteration since such artefacts can be the outcome of multiple datasets and/or AI models, the combination of which, may introduce several challenges from a technical as well as from an IPR perspective.

2.10 Provenance Engine

2.10.1 Description

The Provenance Engine serves the purpose to collect, process and manage 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*. Subsequently this data is to be transformed into RDF-format before forwarding it to the Provenance Information Store API, which handles the ingestion of the data into a triple store. It shall also be possible to retrieve provenance data regarding a specific asset. Code for the Provenance Engine will be provided for the upcoming deliverable 2.3.

2.10.2 API documentation

An early version of a possible API-design can be found in the project's code repository[15]. However, some services may be removed while new ones may be added as the development progresses.

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

Table 2-20: Provenance Engine – API Endpoints Overview

Endpoint	Method	Description
/provenance/{asssetUUID}	POST	Creating a new provenance information entry for the specified asset. Required arguments: • Asset UUID • Version tag • Title • Publishing entity • Source assets • Performed operation • Operation description • Execution entity • Operation date
	GET	Retrieve the latest provenance information of the specified asset. Required arguments: • Asset UUID • Version tag
/lineage/{assetUUID}	GET	Get full lineage of specified asset. Required Arguments: • Asset UUID

2.10.3 User Interface

The component doesn't have a graphical user interface.

2.10.4 Technology stack and License information

A library, which as of now constitutes a suitable foundation for the Provenance Engine is Python prov. It offers a huge variety of provenance data management tools, while being W3C-PROV compliant. It is published under MIT license. The services will be provided via a REST API built based on Python Flask, which is published under BSD3 clause license.

2.10.5 Code repository

This information will be released once the development of the component has progressed.

2.10.6 Pending functionalities roadmap

The table below presents the planned functionalities planned for the next release of the component.

Table 2-21: Provenance Engine – Pending Functionalities

Feature ID	Functionalities & Implementation Status
PE.1	Backend, which converts incoming provenance data (which will likely be of JSON-format) into triple store suitable RDF-format vice versa.
PE.2	API, which offers adding provenance data to the Provenance Information Store as well as retrieving provenance data from it.

2.10.7 Considerations

There are no specific considerations at the moment.

2.11 Access Manager: Policy Engine

2.11.1 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.

Based on the design specifications that were documented in deliverable D2.1, the Policy Engine is a purely backend component hence it provides a rich set of backend operations which are leveraged by the rest of the components via well-defined REST interfaces. The backend operations can be grouped into two core functionalities, the ones related to the formulation of the XMANAI access control model and the ones 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 deliverable D2.1 (**PEN1**). Then, the Policy Engine receives and combines the access control policies, as provided by the Policy Editor in order to be evaluated when a new request is received. In addition to this, 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**). Within this context, the Policy Engine provides the means to constantly deploy, update and enforce any new or updated access policy or a combination of access policies (**PEN.2**) that is received by the Policy Editor. The formulation of the access control decisions is based on the evaluation of all access requests against the defined access control model in order to yield an access control decision that will allow or deny the access to the requested asset (**PEN.4**). To achieve this, the Policy Engine provides a 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 first release of the Policy Engine implements the first set of features that were documented in detail in the deliverable D2.1. The following table provides an overview of the exact status of the implementation in the current release.

Table 2-22: Policy Engine – Implemented Functionalities

Feature ID	Functionalities & Implementation Status
PEN1	**Formulation of the access control model.** In this first 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. It should be noted though that the access control model is bound to change with the addition of more attributes that may be introduced as the project evolves.
PEN2	**Instant deployment, update and enforcement of the access policies.** In this first release, the Policy Engine is able to handle the evolution of access policies by receiving and instantly deploying and enforcing the new or updated access policies as received by the Policy Editor.
PEN3	**Combination and enforcement of multiple policies for an asset based on logical reasoning**. In this first release, the Policy Engine supports the definition of an access policy where multiple conditions can be defined based on logical reasoning.
PEN4	**Regulate the access to any asset of XMANAI**. In this first 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. The evolution of PEN3 will result in an updated implementation of PEN4 in the upcoming release.
PEN5	**Provide a set of well-defined API interfaces for the access control mechanism**. In this first release, the Policy Engine exposes a set of APIs suitable for the reception and evaluation of all access requests. In the upcoming release, the set of APIs will be further extended to accommodate the additional features that will be delivered.

2.11.2API documentation

The Policy Engine provides a rich 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 documentation of the aforementioned endpoints is available via the Swagger documentation library in the project's code repository[16].

Table 2-23: Policy Engine – API Endpoints Overview

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

2.11.3User Interface

[16] https://GitLab.com/xmanai-h2020/api-docs/-/blob/main/Policy%20Manager%20OpenAPI.yaml

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.

2.11.4 Technology stack and License information

As documented in deliverable D2.1, the implementation of the first 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[1], the powerful policy enforcement framework which provides a rich set of functionalities required for the implementation of the access control model.
- jCasbin[2], the Java-based library of Casbin which supports the implementation of the access control mechanism based on Casbin.
- Spring Boot framework[3], 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 11 programming language.

In terms of licensing, the Policy Engine component is closed source. The developed version will be 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[17], access to which can be provided by a request sent to the project coordinator.

2.11.6 Pending functionalities roadmap

In the upcoming version of the Policy Engine, the handling and interpretation of the access policies provided by the Policy Editor will be expanded to optimise the formulation process of the access control model with the ability for the owner of the asset to define even more fine-grained access policies with additional attributes (**PEN1**). In addition to this, the set of provided API interfaces will be also expanded in order to expose the new capabilities of the Policy Engine (**PEN5**).

2.11.7Considerations

At the time of writing, no challenges or limitations are foreseen for the development of the next version of the Policy Engine.

2.12 Access Manager: Policy Editor

2.12.1Description

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

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 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 will offer a novel and user-friendly user-interface through which the owner of the asset will be able to perform all the access policy lifecycle management operations.

Based on the design specifications that were documented in deliverable D2.1, the Policy Editor offers a rich set of features that can be conceptually organised into the access policy lifecycle management operations and the respective user-interface and the interaction with the Policy Engine in order to provide the required input for the formulation of the access control model. Hence, the Policy Editor provides the means to define, store, update and delete access policies for any protected asset of the XMANAI platform with the respective backend (**PED.1**) and frontend functionalities (**PED.4**). In addition to this, the Policy Editor facilitates the reuse of existing policies on a different asset (**PED.2**) and the definition of simple 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 and restrictions imposed by the access control model's implementation as set by the Policy Engine (**PED.5**).

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

Table 2-24: Policy Editor – Implemented Functionalities

Feature ID	Functionalities & Implementation Status
PED.1	**Enable the flexible definition, storage, update and deletion of access policies.** In this first release, the Policy Editor facilitates all the CRUD operations for access policies. As the project evolves, access policy definition may be enhanced with additional attributes, hence the implementation will be properly optimised in that case.
PED.2	**Facilitate the reuse of previous defined access policies.** In this first release, the Policy Editor enables the reuse of existing policies for a different asset of the same owner.
PED.3	**Enable the combination of the multiple access policies using Boolean logic**. In this first release, the Policy Editor provides the means to define complex access policies with a combination of multiple conditions with use of logical reasoning.
PED.4	**Provide the easy-to-use and novel user-interface for all the access policy lifecycle management operations**. The implementation of the specific functionality has not started in the current release and it is planned for the upcoming release.
PED.5	**Enable the loading of the access policies into the Policy Engine.** In this first release, the Policy Editor facilitates the loading of the defined access policies 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 first release, the Policy Editor facilitates the immediate update of the access control model by interacting with the Policy Engine.

2.12.2API documentation

The Policy Editor provides a set of well-defined APIs that interconnect the backend and frontend operations of the component. The documentation of the aforementioned endpoints is available via the Swagger documentation library in the project's code repository[18]:

Table 2-25: Policy Editor – API Endpoints Overview

Endpoint	Method	Description
/api/v1/editor	POST	Adds a new set of rules for a specific asset.
/api/v1/editor	GET	Returns the existing rules for a specific asset.
/api/v1/editor	PUT	Updates the existing rules for a specific asset.

2.12.3User Interface

The user interface of the specific component (PED.3) is planned for the upcoming release.

2.12.4Technology stack and License information

As documented in deliverable D2.1, for the implementation of the first version of the Policy Editor the same list of frameworks, libraries and technologies as for the Policy Engine is leveraged. 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 the 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 will be available through the XMANAI platform.

2.12.5Code 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[19], access to which can be provided by a request sent to the project coordinator.

2.12.6Pending functionalities roadmap

[18] https://GitLab.com/xmanai-h2020/api-docs/-/blob/main/Policy%20Manager%20OpenAPI.yaml
[19] https://GitLab.com/xmanai-h2020

In the upcoming version of the Policy Editor, the major update will be the implementation of the user-friendly user-interface of the component (PED.4) which will enable the owners of the assets to perform all the access policy lifecycle management operations. Besides this, the component will be enhanced with the ability to define even more fine-grained access policies with additional attributes (PED1).

2.12.7 Considerations

At the time of writing, no challenges or limitations are foreseen for the development of the next version of the Policy Engine.

2.13 Identity & Authorisation Management

2.13.1 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.

Based on the design specifications that were documented in deliverable D2.1, the Identity and Authorisation Manager provides the 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 required 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 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. To this end, significant effort has been invested in order to implement 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. Nevertheless, optimisations and enhancements will be introduced based on the evolution of the project. The following table provides an overview of the exact status of the implementation in the current release.

Table 2-26: Identity & Authorisation Management – Implemented Functionalities

Feature ID	Functionalities & Implementation Status
IAM.01	**Complete user management lifecycle by implementing solid organization registration**. In this first release, the Identity and Authorisation Manager implements the mechanism for the registration and verification of an organisation, as well as the update of the organisation's profile. In the upcoming release, the user interface related to the aforementioned mechanism will be implemented.

Feature ID	Functionalities & Implementation Status
IAM.02	**Enable the invitation and registration of users under a single organization**. In this first release, the Identity and Authorisation Manager implements the user's invitation and registration process as well as all the CRUD operations on the user's profile. In the upcoming release, the user interface related to the aforementioned mechanism will be implemented.
IAM.03	**The authentication mechanism for the platform's services and offerings**. In this first release, the Identity and Authorisation Manager implements the mechanism that verifies and controls the access via a login mechanism that verifies the provided credentials.
IAM.04	**Regulate the intercommunication of the various components of the platform**. In this first release, the Identity and Authorisation Manager implements the authorisation mechanism which controls the intercommunication level of the components on either a service or an endpoint level.

2.13.2API documentation

The Identity and Authorisation Manager provides a rich 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 complete documentation of the aforementioned endpoints is available via the Swagger documentation library in the project's code repository[20].

The most import endpoints are also available in the following table:

Table 2-27: Identity & Authorisation Management – API Endpoints Overview

Endpoint	Method	Description
/api/v1/organization	POST	Request for a new organization in the platform.
/api/v1/organization	GET	Shows a list of all the organization registration requests.
/api/v1/organization/{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.

[20]https://GitLab.com/xmanai-h2020/api-docs/-/blob/main/Identity%20Authorization%20Manager%20OpenAPI.yaml

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	Verifies the identity of the user / checks if user has a valid JWT.
/api/v1/account	GET	Returns user's account details.
/api/v1/group/mine	GET	Returns the organization that the user belongs to.

2.13.3 User Interface

As explained in Table 2-26, the user interface of the specific component (IAM.1 and IAM.2) is planned for the upcoming release.

2.13.4 Technology stack and License information

As documented in deliverable D2.1, the implementation of the first 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[4], 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[5], the well-established reactive-stack web framework of Spring which is suitable for non-blocking APIs and web stack solutions.
- Spring Security[6], the well-established powerful and highly customizable authentication and access-control framework of Spring.
- R2DBC[7], the most suitable database driver for Reactive designed from the ground up for reactive programming with SQL databases.
- PostgreSQL[8], 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[9], the widely used key-value store that provides fast processing and easy data structure representation.
- MinIO[10], 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[11], 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.

2.13.5 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[21], access to which can be provided by a request sent to the project coordinator.

2.13.6 Pending functionalities roadmap

As explained in section 2.13.1, the current release of the Identity and Authorisation Manager implements the complete set of features that were documented in detail in the deliverable D2.1. The focus in the upcoming release will be on the user-interface aspects of the component (IAM.1 and IAM.2), as well as the incorporation of enhancements and updates based on the needs of the project.

2.13.7 Considerations

At the time of writing, no challenges or limitations are foreseen for the development of the next version of the Identity and Authorisation Manager.

2.14 Anonymiser

In this section, we describe the development of the Anonymiser.

2.14.1 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[22], 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

[21] https://GitLab.com/xmanai-h2020
[22] https://amnesia.openaire.eu/index.html

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 user needs can be inspected and applied with simple clicks. The anonymized data can then be saved locally.

Table 2-28: Anonymiser – Available Functionalities

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

2.14.2 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.

2.14.3 User Interface

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

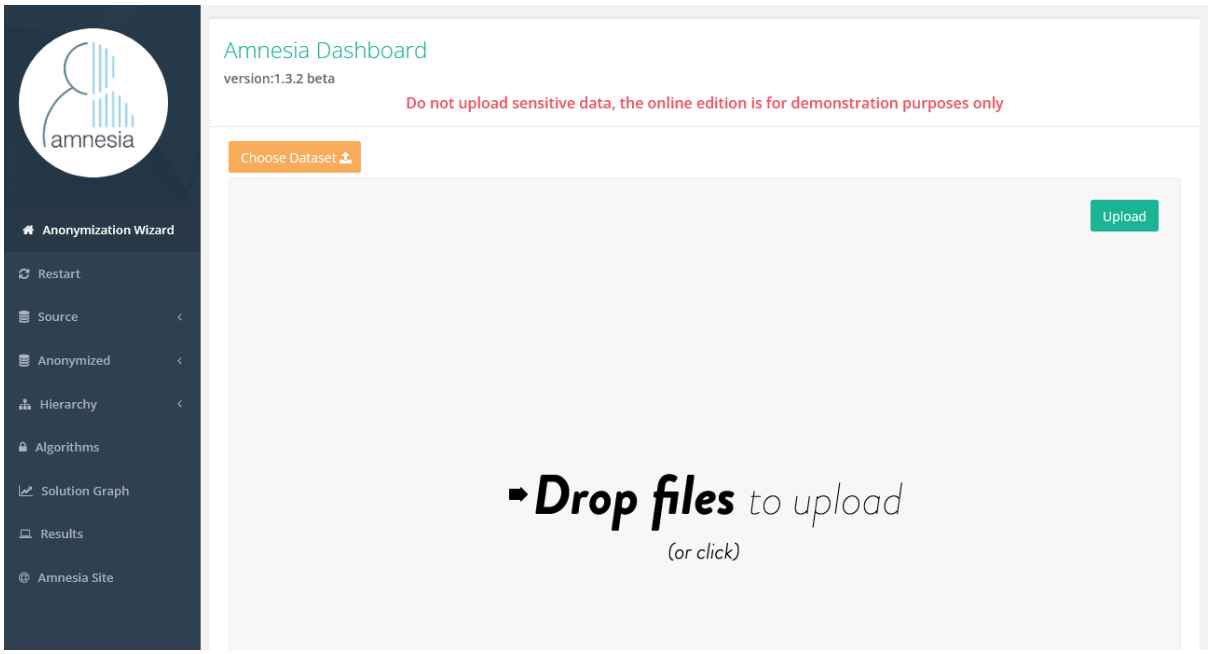

Figure 2-15: The interface for uploading a tabular dataset in order to anonymize it

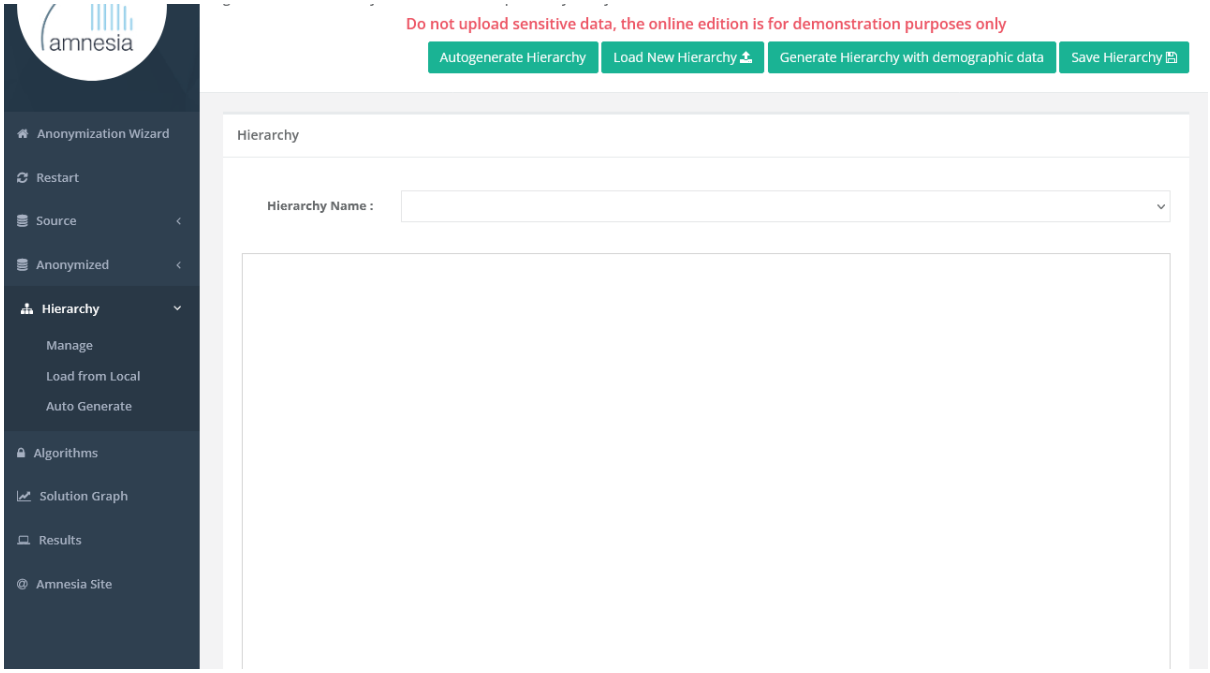

Figure 2-16: The interface for generating an hierarchy, before applying an anonymization algorithm.

Figure 2-17: The interface for applying an anonymization algorithm.

2.14.4Technology 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.

2.14.5Code repository

The source code for the Amnesia tool is managed in the GitHub repository[23].

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

2.14.6 Pending functionalities roadmap

There are no pending functionalities.

2.14.7 Considerations

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

3 Conclusions and Next Steps

The current deliverable D2.2 “XMANAI Asset Management Bundles – First Release” presents the first release of the components responsible in XMANAI for assets management and previously designed in the deliverable D2.1 “Asset Management Bundles Methods and System Designs”. D2.2 is a result of the coordinated work of in all tasks of the work package 2. The deliverable provides detailed technical information for each of the 13 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.
- The list of functionalities planned for the next final release of the AI Bundles.
- Assumptions made during the development of the first release of the XMANAI Asset Management Bundles, as well as any possible challenges that will need to be addressed in the future.

The next steps include the integration of the XMANAI Asset Management Bundles developed in WP2 with the other services implemented in WP3 and WP4 under the first release of the XMANAI platform that will be reported in deliverable D5.2 “XMANAI Platform – Alpha Version” on M21. The development of XMANAI Asset Management Bundles will continue, in order to provide additional features, as well as modifications and improvements of the existing component functionalities, based on the roadmap presented in the current document and the new requirements that might occur in the future, through the feedback provided by the demo partners during the activities of WP6. The next release of the WP2 XMANAI Asset Management Bundles will follow on M30 that will contain the pending features and improvements of the existing ones and will be documented in D2.3 “XMANAI Asset Management Bundles – Second Release”.

References

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

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

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

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

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

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