Promoting and Incentivising Federated, Trusted, and Fair Sharing and Trading of Interoperable Data ASsets

D2.3
Data Management and Protection services - Beta version

Editor(s)	Yury Glikman
Lead Beneficiary	FHG
Status	Final
Version	1.0
Due Date	30/06/2024
Delivery Date	30/06/2025
Dissemination Level	PU

Funded by the European Union under Grant Agreement 101093016. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Project	PISTIS – 101093016
Work Package	WP2 - PISTIS Data Spaces Factory and Trusted Data Management Services
Deliverable	D2.3 Data Management and Protection services - Beta version
Contributor(s)	Suite5, ATOS, FHG, EUT, SPH, UBITECH, ATHENA, ASSENTIAN, ICCS
Reviewer(s)	Ilia Christantoni, Dimitra Tsakanika (DAEM), Florian Feik (TRAFF)
Abstract	This deliverable presents the Beta release of the core PISTIS factory services dealing with data management and protection, and of the Data Explorer Service

Executive Summary

PISTIS aims to develop a reference platform for the sharing/trading and monetisation of the proprietary data of an organization, guaranteeing a secure, trusted and controlled exchange and usage of data assets and data-driven intelligence.

This document presents the Beta version of the design and implementation of the PISTIS components belonging to one of the following bundles from the PISTIS architecture, as coming out of the design and development activities of WP2 of the project:

- **Data Management and Assessment bundle**, that is responsible for the collection of data from existing repositories available to an organisation, the refinement, transformation, and improvement of data, judging also its quality and providing services to improve it and make it interoperable.
- **Data & Metadata Storage bundle**, that is delivering a catalogue for the data available for each organisation and those that are made available as "published" data over the whole ecosystem, alongside with the appropriate data storage facilities to hold the data.
- **Data Discovery bundle**, that provides services for searching and discovering the available data assets that might be of interested to a Data Consumer
- **Data Exchange bundle**, that facilitates the peer-to-peer exchange of the data assets between a Data provider and a Data Consumer, adhering to the terms of the contract that has been signed to govern the overall transaction.
- **Security, Trust & Privacy Preservation bundle**, that is offering services for strengthening data security and privacy.
- **AI & Interoperability Repos bundle**, that provides the different repositories for storing and propagating different models (data models, AI models and metadata models) that need to be consumed by the various components.

The Beta version of the components presented in this document represents an evolution of the Alpha version previously documented in Deliverable D2.2, "Data Management and Protection Services – Alpha Version." This document builds upon D2.2 by updating existing sections and introducing new sub-sections for each component, including:

- Main Improvements in the Beta Version
- Developer Documentation
- Source Code

Most of the components described herein are open source and publicly accessible. The remaining components are maintained in private repositories, with access restricted to the project team and reviewers.

As a next step, the consortium will integrate the components presented in this deliverable and in Deliverable D3.3 into the Beta release of the PISTIS platform, as documented in Deliverable D4.3.

Table of Contents

List of Figures

Terms and Abbreviations

ABAC	Attribute Based Access Control
ABE	Attribute Based Encryption
ADB	Asser Description Bundle
AI	Artificial Intelligence
API	Application Programming Interface
CF	Collaborative filtering
CRUD	Create, Read, Update, Delete
DCAT	Data Catalogue Vocabulary
DLT	Distributed Ledger Technology
DNN	Deep neural networks
DoA	Description of Action
DSE	Dynamic Searchable Encryption
DVDs	Data Value Dimensions
DVS	Data Valuation Service
eIDAS2	electronic IDentification, Authentication and trust Services 2
FAIR	Findable, Accessible, Interoperable, Reusable
FTP	File Transfer Protocol
GDPR	General Data protection Regulation
GNN	Graph Neural Networks
HTTP	HyperText Transfer Protocol
ID	Identity
IDS	International Data Spaces
IOTA	Internet of Things Application
JSON	JavaScript Object Notation
JWT	JSON Web Token
kNN	k-Nearest Neighbour
LSH	Locality-Sensitive Hashing
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
MVP	Minimum Viable Product
OIDC	OpenID Connect
PROV	Provenance
RBAC	Role Based Access Control
RDF	Resource Description Framework
REST	Representational state transfer
SE	Searchable Encryption
SQL	Structured Query Language
SSI	Self-Sovereign Identity
ToC	Table of Contents
UUID	Universal Unique Identifier
WP	Work Package
XAI	eXplainable AI
YAML	Yet Another Modelling Language

1 INTRODUCTION

The PISTIS project adopts an agile development methodology, progressing through two iterations of design, implementation, and evaluation. In the initial phase, technical requirements and user stories were gathered (D1.2), and relevant methods, technologies, models, and specifications for addressing them were analysed (D2.1, D3.1). The initial architecture of the PISTIS Platform—illustrating the individual components and their interrelationships—was then defined in Deliverable D4.1. Following this, the first (Alpha) version of the core PISTIS factory components, focusing on data discovery, management, and protection, was presented in Deliverable D3.2.

This document builds on D3.2 and presents the subsequent Beta version of these components, which have been refined according to the development plan (backlog). It accompanies the implementation by detailing each component's purpose, requirements, architecture, technologies used, and user interface (where applicable). In addition, the document highlights the enhancements made since the Alpha release and provides information about the availability of source code and developer documentation. The components compose the Data Management and Assessment, Data & Metadata Storage, Data Discovery, Data Exchange, Security, Trust & Privacy Preservation and AI & Interoperability Repos bundles of the PISTIS architecture shown in the Figure below.

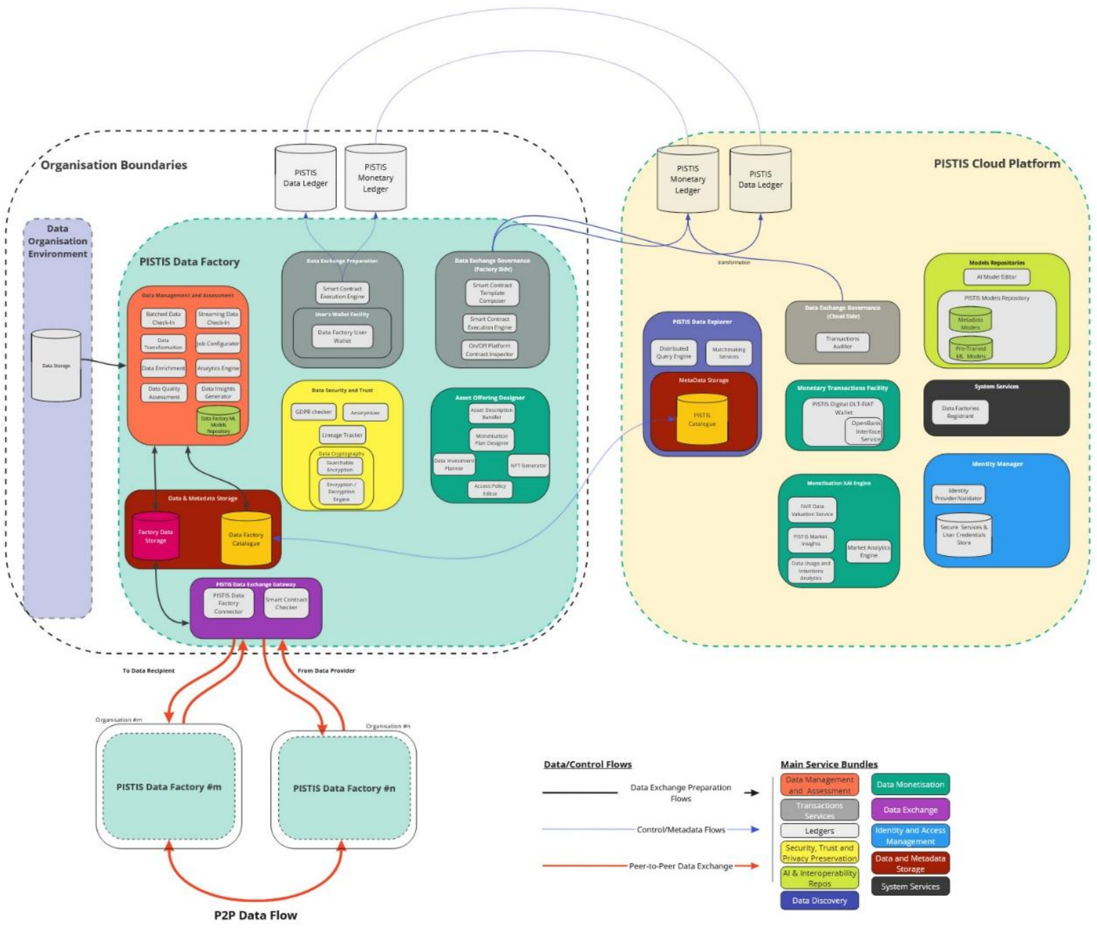


Figure 1: PISTIS Architecture

The Beta version of another set of PISTIS components, belonging to the **Data Monetisation, Transaction Services, and Ledgers bundles**, is presented in Deliverable D3.3.

Both sets of components will be integrated into the Beta release of the PISTIS Platform (D4.2), marking the completion of the second design and development iteration. This will be followed by the final (Version 1.0) iteration of the development process.

The source code for the PISTIS components is maintained in the project's GitHub repository. Access can be granted upon request:

https://github.com/orgs/PISTIS-Platform/

1.1 DOCUMENT STRUCTURE

The document is structured as follows:

Section 1 is the introduction and this document structure description.

Sections 2 - 7 describe the different bundles and present details on the different components using the same structure of the subsections:

- Component Description
- Main Improvements in Beta Version
- Component Backlog
- Functional Requirements
- Non-Functional Requirements
- Component Architecture
- Technology Background
- Graphical User Interface
- Developer Documentation
- Source Code.

Finally, Section 8 concludes the document.

2 DATA MANAGEMENT AND ASSESSMENT BUNDLE

The Data Management and Assessment bundle is responsible for the collection of data from existing repositories available to an organisation, the refinement, transformation, and improvement of data, judging also their quality and providing services to improve them and make them interoperable.

This bundle consists of the following components:

- Data Check-In
- Data Transformation
- Job Configurator
- Analytics Engine
- Data Enrichment
- Data Quality Assessment
- Data Insights Generator

These are presented in the following sub-sections.

2.1 DATA CHECK-IN

Data Check-In enables support for data sources that will supply data for the solution workflow, as data sources can come in a variety of forms (repositories, data streaming flows, and so on). The Alpha version of Data Check-In presented in the deliverable D2.2 was supporting only integration of batched data. In the Beta version the batched data check-in got further improvements, which are presented in the next sub-section 2.1.1. In addition to that, a new Data Check-In sub-component for streaming data was added. It is presented in the sub-section 2.1.2.

Figure 2 depicts an overview of the data injection component, highlighting the two primary submodules that handle batch (sub-section 2.1.1) and streaming data (sub-section 2.1.2).

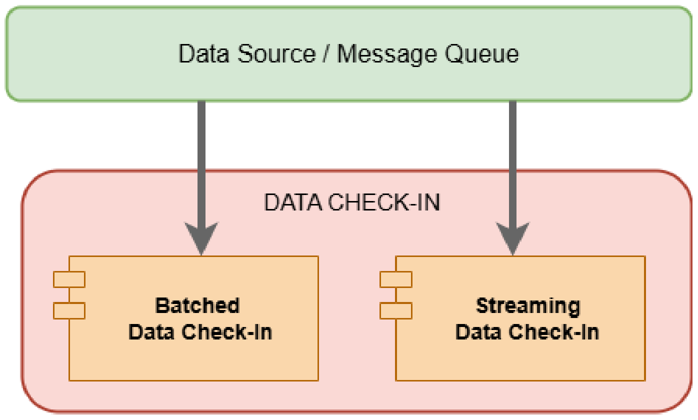


Figure 2. Data Check-In Sub-modules.

2.1.1 Batched Data Check-in

2.1.1.1 Component Description

Batched Data Check-In serves as input for the whole data workflow in the PISTIS platform allowing several ways for data ingestion which must include the followings:

- File upload
- FTP Server
- API

Batched Data Check-In offering in terms of functionalities is detailed below:

- *UploadData*: This functionality should allow the end user to provide a data file to be stored in a server to be consumed by the subsequent data processing workflow defined in the PISTIS platform. Some basic verifications could be carried out in order to check some requirements regarding the data file provided (e.g. size limits, data formats, etc.).
- *GetDataFromFTP*: In case the data to be ingested by the workflow is stored in an FTP server, a method will be provided to retrieve that data. This method should be called providing all the information needed to get access to the data (I.e. endpoint, path to the file, filename, required credentials, etc.).
- *GetDataFromAPI*: In this case, the data is retrieved from a defined API or from other Data Space Connectors. The connectors should be compatible with GAIA-X and IDS. It will be required, as in the previous case, to get all the details (as well as credentials when necessary) needed in order to get access to the required data source.

2.1.1.2 Main Improvements in Beta Version

The following improvements have been introduced in beta version:

- Extended file upload formats include CSV, TSV, XML, XLSX, JSON, and Parquet.
- Automatic dataset encoding identification and export to UTF-8 format. This feature was necessary for certain of our pilots who deal with datasets encoded with Windows-1253 (Greek).
- Get data from FTP Server: To connect to an FTP server via FTP data upload, give the following information:
  - The IP address or hostname of your FTP server might be 10.1.1.58 or ftp.company_name.com.
  - Port number: The TCP port that your FTP server listens on. Normally, this would be port 21.
  - Login credentials: This contains your login and password. This information will be used to authenticate users on your FTP server.
  - File path refers to the relative location where files are stored on an FTP server.

- Get data from API: The fundamental principles of employing an HTTP API in PISTIS context are straightforward:

- Make an HTTP request to the API's URLs, perhaps supplying authentication data (such an API key) to demonstrate our authorisation.
- Get back the data
- Store physically the data using the factory data storage component.

2.1.1.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority[1]	Acceptance Criteria	Status[2]	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data Provider	Upload a file into Pistis ecosystem.	Have data available in Pistis ecosystem	Alpha	Data stored properly in Factory Data Storage.	Done	PISTIS.OUS.01
UC_02	Data Provider	Inject Data coming from an FTP Server.	Have data available in Pistis ecosystem	Beta	Data stored properly in Factory Data Storage.	Done	PISTIS.OUS.01
UC_03	Data Provider	Inject Data coming from a Data Space (API)	Have data available in Pistis ecosystem	Beta	Data stored properly in Factory Data Storage.	Done	PISTIS.OUS.01
UC_04	Data Provider	Manage Data Check-In from GUI	Have data available in Pistis ecosystem	Beta	Data stored properly in Factory Data Storage.	Done	PISTIS.OUS.01

2.1.1.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	PISTIS supports data injection/registration from different data source type.	UC_01, UC_02, UC_03, UC_04	PISTIS platform must support multiple type of data sources such as FTP, HTTP APIs, SFTP, DB connections, etc.
FR_02	PISTIS supports various format and description languages of metadata.	UC_01, UC_02, UC_03, UC_04	PISTIS platform must support various format (JSON, XML, RDF, etc.) and various description languages or standards or ontologies for the metadata.

2.1.1.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Security	NFR1	PISTIS ensures that only authorised users can register datasets.

[1] Priority based on the releases: Alpha / Beta / v1.00

[2] Upcoming / In Progress / Done (delivered in alpha/beta/v1.00) / Obsolete

Compatibility	NFR2	The component must be able to accept the input data with Greek encoding as Windows-1253.

2.1.1.6 Component Architecture

The Data Validation and Checking Module (DACM) and the Data Injection Module (DIM) are the two primary modules of the Data Check-In component, as depicted in Figure 3, which also displays the internal architecture of the component.

The business logic found in the DACM will enable:

1) Perform some basic verifications to confirm that the data file requirements—such as size limitations and data formats—are met, and
2) Verify that all the information required to ensure access to the data source has been supplied, including any access credentials that may be required.

After the DACM has verified and guaranteed access to the data, the DIM will be responsible for ingesting them. DIM will include dedicated submodules for every kind of ingestion needed by PISTIS, including FTP, file uploads, and API retrieval.

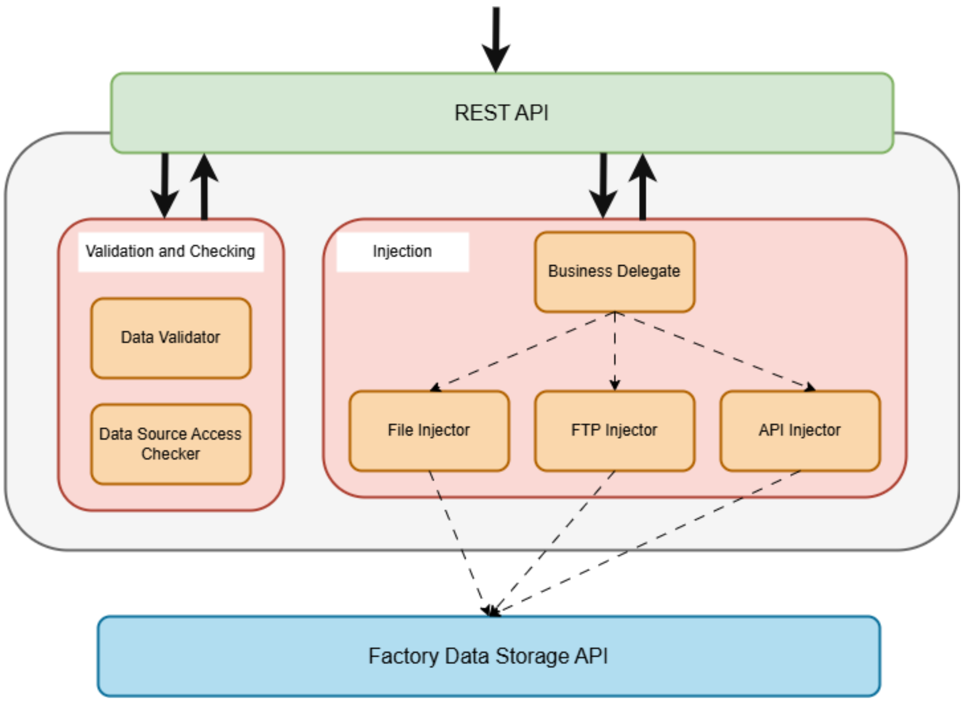


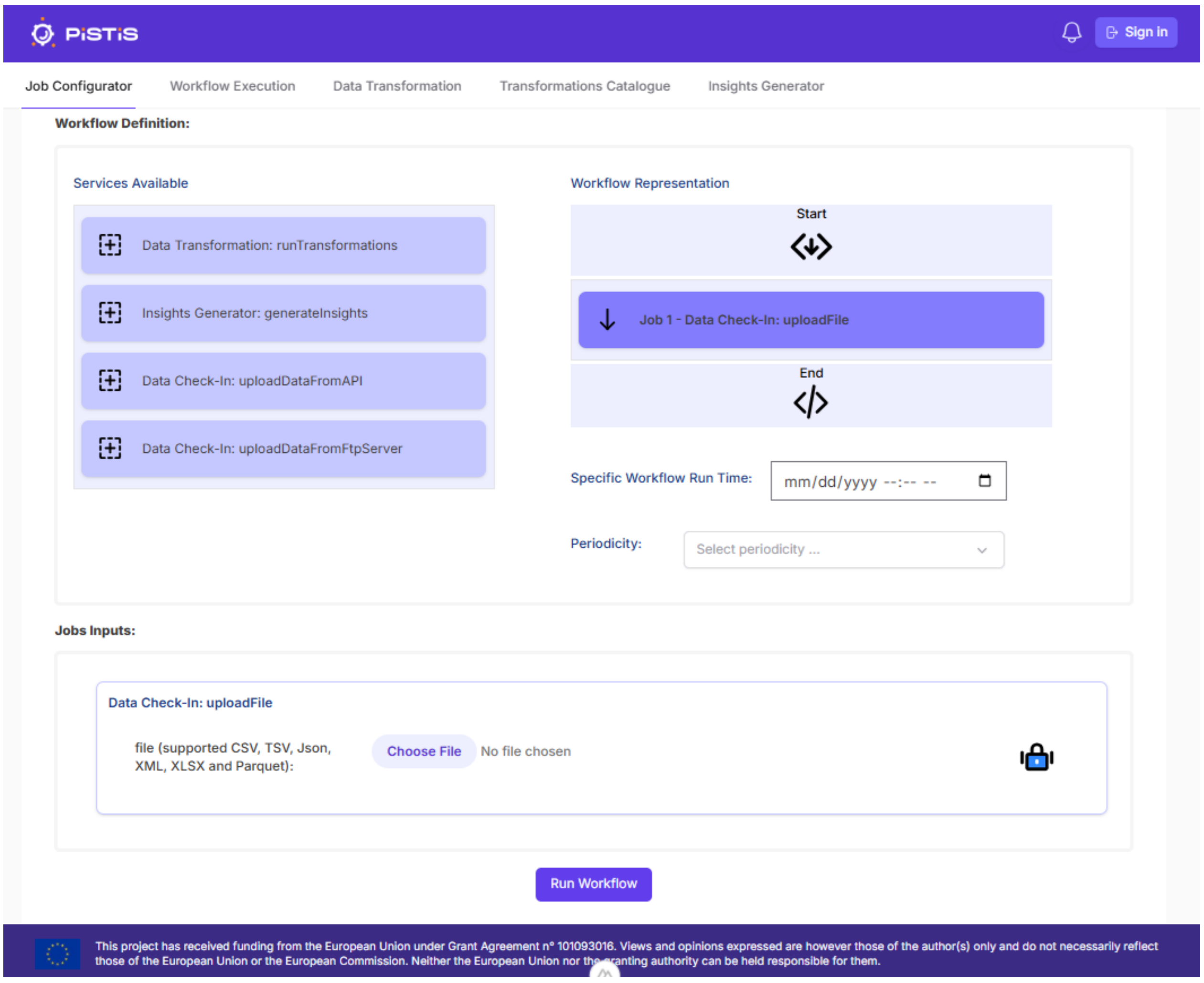
Figure 3: Batched Data Check-In Architecture

2.1.1.7 Technology Background

Batched Data Check-In is built on Python and uses commonly used data transport modules such as HTTP, FTP, and request, among others.

2.1.1.8 Graphical User Interface

The Data Check-In GUI has been included into the Job Configurator GUI for the Beta version. A screenshot of the File Upload GUI is given in Figure 4.



Workflow Definition:

Services Available

- Data Transformation: runTransformations
- Insights Generator: generateInsights
- Data Check-In: uploadDataFromAPI
- Data Check-In: uploadDataFromFtpServer

Workflow Representation

Start

Job 1 - Data Check-In: uploadFile

End

Specific Workflow Run Time: mm/dd/yyyy --:-- --

Periodicity: Select periodicity ...

Jobs Inputs:

Data Check-In: uploadFile

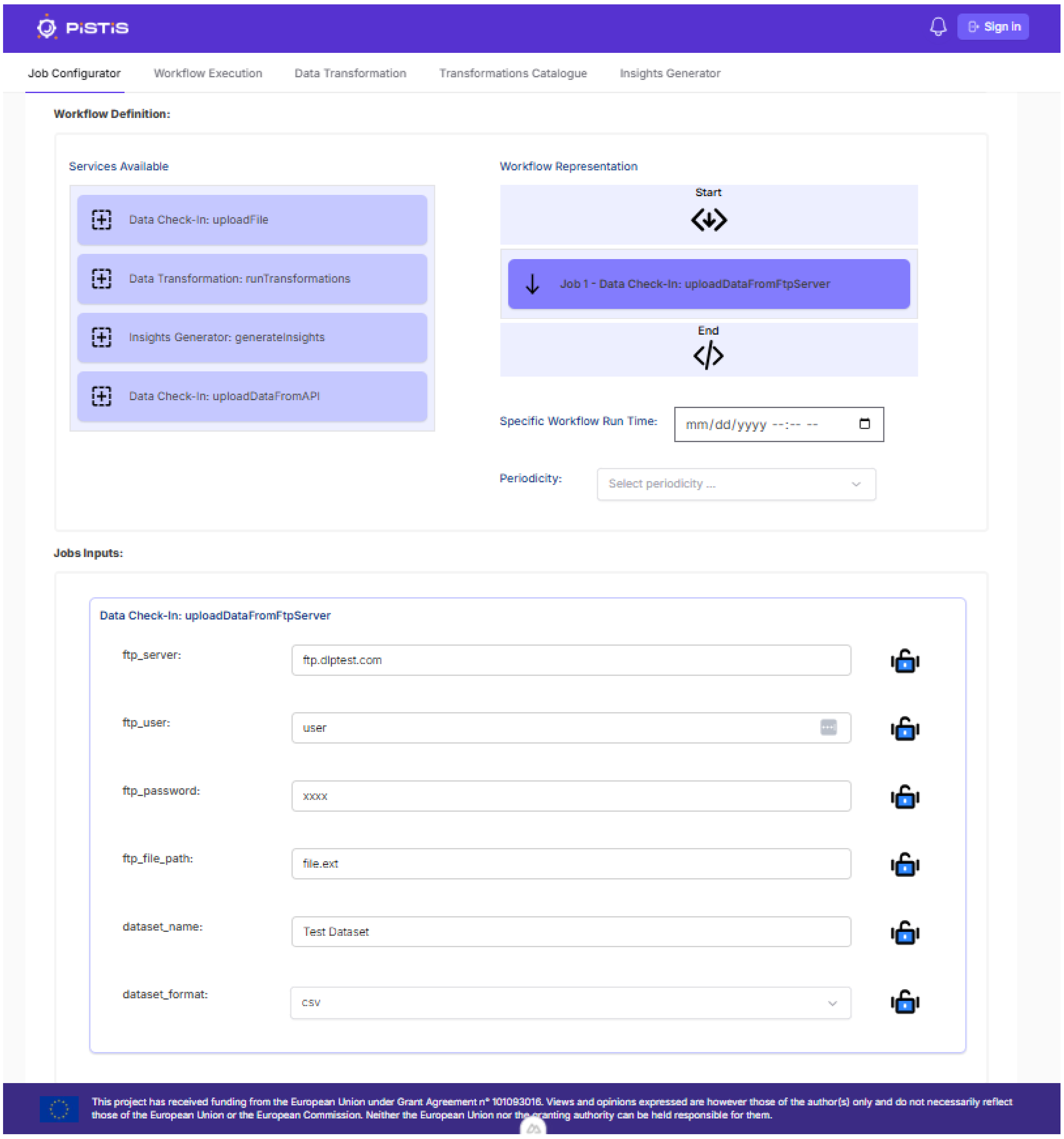
file (supported CSV, TSV, Json, XML, XLSX and Parquet): Choose File No file chosen

Run Workflow

This project has received funding from the European Union under Grant Agreement n° 101093016. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Figure 4: File upload GUI

Figure 5 depicts the GUI that supports data ingestion from an FTP server.



Workflow Definition:

Services Available

- Data Check-In: uploadFile
- Data Transformation: runTransformations
- Insights Generator: generateInsights
- Data Check-In: uploadDataFromAPI

Workflow Representation

Start

Job 1 - Data Check-In: uploadDataFromFtpServer

End

Specific Workflow Run Time: mm/dd/yyyy --:-- --

Periodicity: Select periodicity ...

Jobs Inputs:

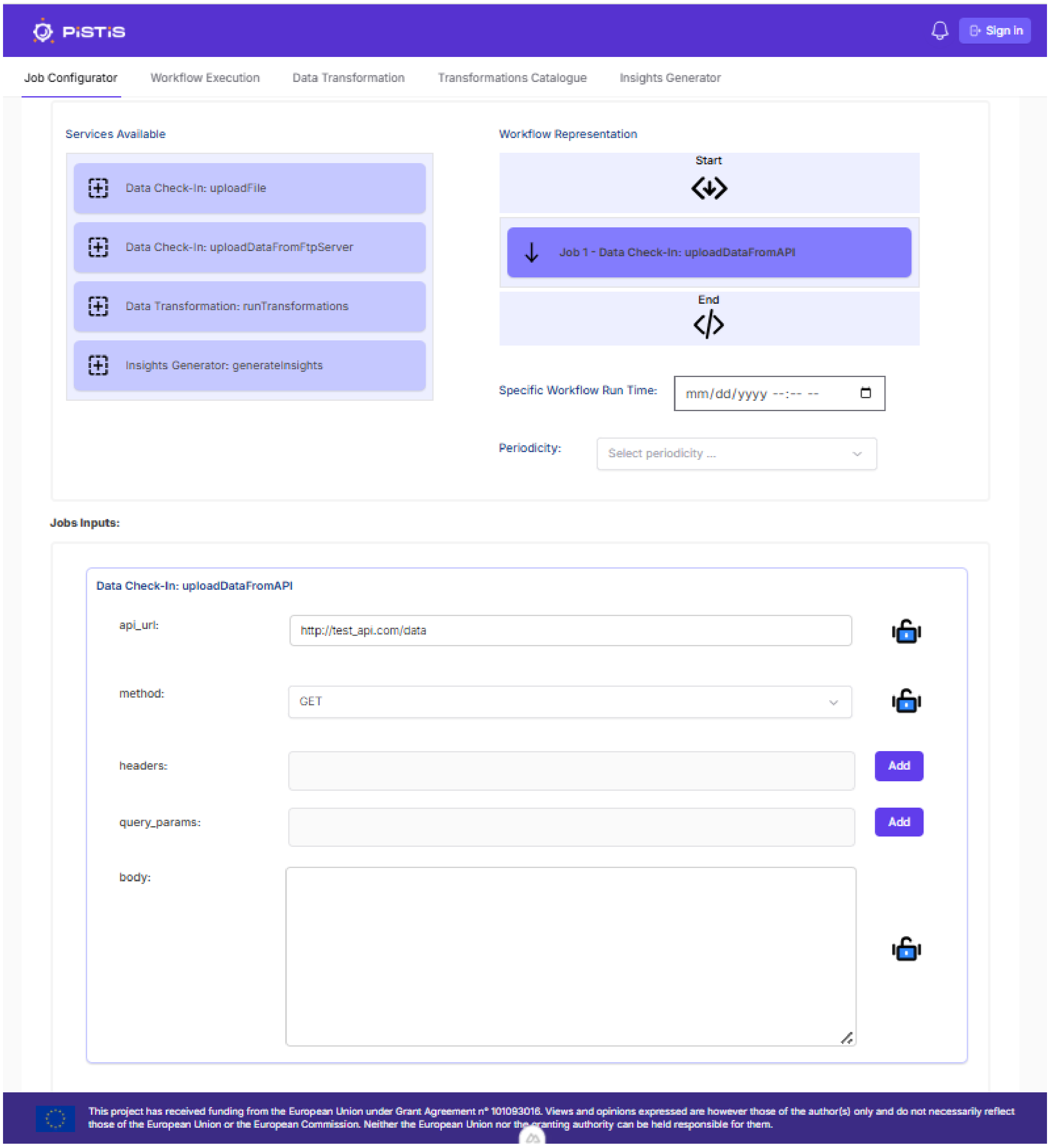
Data Check-In: uploadDataFromFtpServer

ftp_server:	ftp.dlptest.com
ftp_user:	user
ftp_password:	xxxx
ftp_file_path:	file.ext
dataset_name:	Test Dataset
dataset_format:	csv

This project has received funding from the European Union under Grant Agreement n° 101093016. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Figure 5: FTP data upload GUI

Finally, Figure 6 illustrates the GUI for obtaining data from API.



Services Available

- Data Check-In: uploadFile
- Data Check-In: uploadDataFromFtpServer
- Data Transformation: runTransformations
- Insights Generator: generateInsights

Workflow Representation

Job 1 - Data Check-In: uploadDataFromAPI

Specific Workflow Run Time: mm/dd/yyyy --:-- --

Periodicity: Select periodicity ...

Jobs Inputs:

Data Check-In: uploadDataFromAPI

api_url:	http://test_api.com/data
method:	GET
headers:	
query_params:	
body:	

This project has received funding from the European Union under Grant Agreement n° 101093016. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Figure 6: Data fetching from API GUI

2.1.1.9 Developer Documentation

The developer documentation for the Batched Data Check-In component may be accessed on the official PISTIS project documentation site at the following link: https://docs.pistis-market.eu/developers/data-check-in/data-check-in

2.1.1.10 Source Code

The source code developed for the Batched Data Check-In component is currently stored under an internal repository in ATOS premises until the license to be applied to it is decided.

The component's UI (frontend) is part of the platform-frontend repository and its code is maintained in GitHub under the PISTIS project and can be accessed via the respective link: https://github.com/PISTIS-Platform/platform-frontend

2.1.2 Streaming Data Check-in

2.1.2.1 Component Description

The Streaming Data Check-in enables Data Provider to register their data streams to their PISTIS Data Factory which will allow them to make them available to prospective Data Consumers who aim to get access to these streams.

The Streaming data check-in functionality is supporting Kafka streams. This functionality allows users to send data to a Kafka server that is running in the Data Factory of each user and that then holds the data that data consumers might want to get access to.

As such, this component, as part of the overall Data Check-In functionality is offering the following functionality, which has been introduced conceptually in the Alpha version:

- *GetDataFromSubscription*: Create a Kafka topic for each streaming dataset, allowing Data Providers to bring into their Data Factories data from this kind of data source. By means of these, data can be consumed following a given criteria (e.g. defining a time window, a data limit, etc.) and then set for its processing.

2.1.2.2 Main Improvements in Beta Version

This is a component that is introduced for the first time in the beta version.

2.1.2.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority[3]	Acceptance Criteria	Status[4]	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_05	Data Provider	Inject Data coming from subscription (kafka, MQTT)	Have data available in Pistis ecosystem	Beta	Data stored properly in Factory Data Storage.	Done	PISTIS.OUS.01
UC_06	Data Provider	Manage Data Check-In from GUI	Have data available in Pistis ecosystem	Beta	Data stored properly in Factory Data Storage.	Done	PISTIS.OUS.01

2.1.2.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	PISTIS supports data injection/registration from different data source type.	UC_01, UC_02, UC_03, UC_04	PISTIS platform must support multiple type of data sources such as FTP, HTTP APIs, SFTP, DB connections, etc.

[3] Priority based on the releases: Alpha / Beta / v1.00

[4] Upcoming / In Progress / Done (delivered in alpha/beta/v1.00) / Obsolete

2.1.2.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Security	NFR1	PISTIS ensures that only authorised users can register datasets.

2.1.2.6 Component Architecture

The Streaming Data Check-In component is responsible for creating streaming data pipelines and transferring streaming data using Kafka. The component is responsible for registering the data stream where the user (provider) will be able to send (produce) messages to a Kafka topic using the given user credentials for authentication.

The Data Provider Service accepts the data stream information, requests the factory information from the Data Factory storage component such as the factory name and it is responsible for constructing the streaming metadata to be registered in Factory Catalogue component using the internal module of component's Metadata Repository Service.

The Kafka service is responsible for creating the Kafka topic and generating user credentials. These credentials, along with the topic information, are returned to the Factory UI component. The provider user will use them to authenticate with Kafka and send messages to the specified topic.

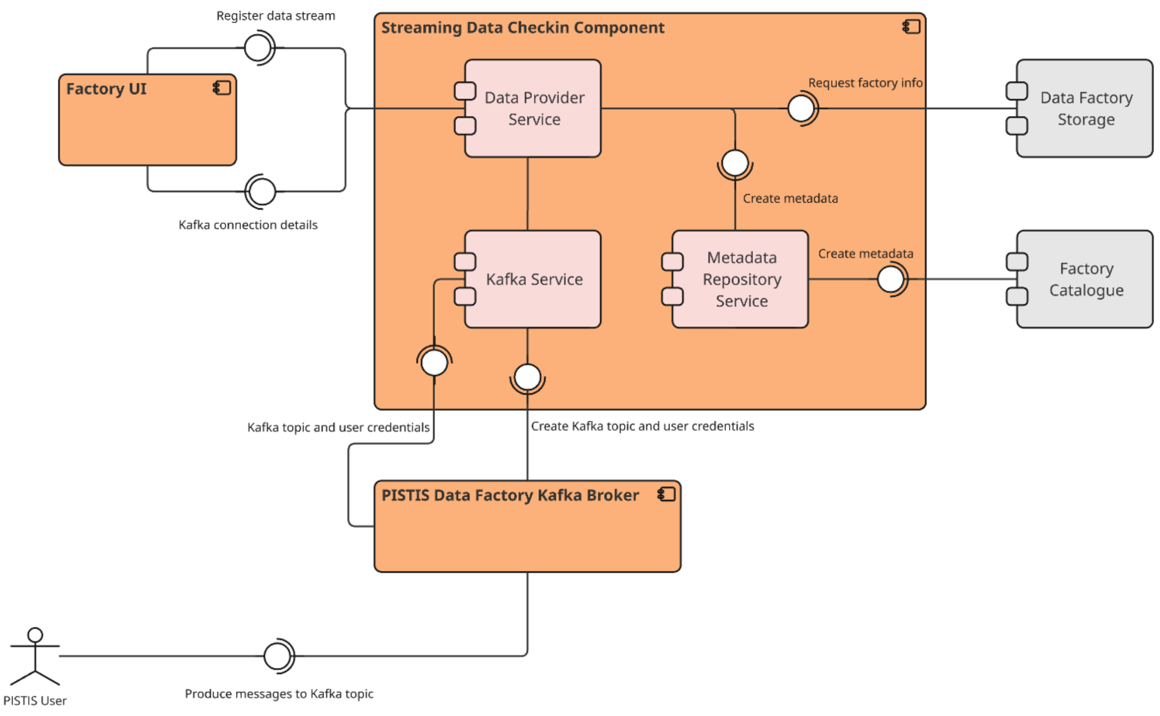

Register data stream
Streaming Data Checkin Component
Factory UI
Data Provider Service
Request factory info
Data Factory Storage
Create metadata
Kafka connection details
Kafka Service
Metadata Repository Service
Create metadata
Factory Catalogue
Kafka topic and user credentials
Create Kafka topic and user credentials
PISTIS Data Factory Kafka Broker
PISTIS User
Produce messages to Kafka topic

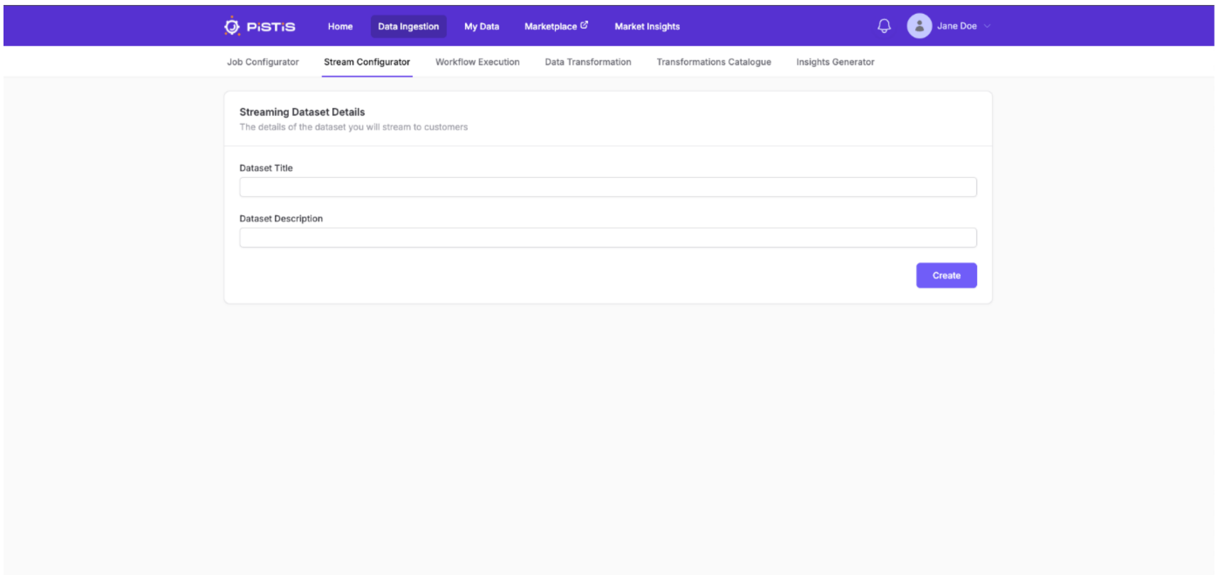
Figure 7: Streaming Data Check-In Architecture

2.1.2.7 *Technology Background*

The main technology used for the Component is Apache Kafka.

2.1.2.8 *Graphical User Interface*

The UI of this component is shown in the following figures. The Data Provider initially provides a title for the streaming data and details about what this contains.



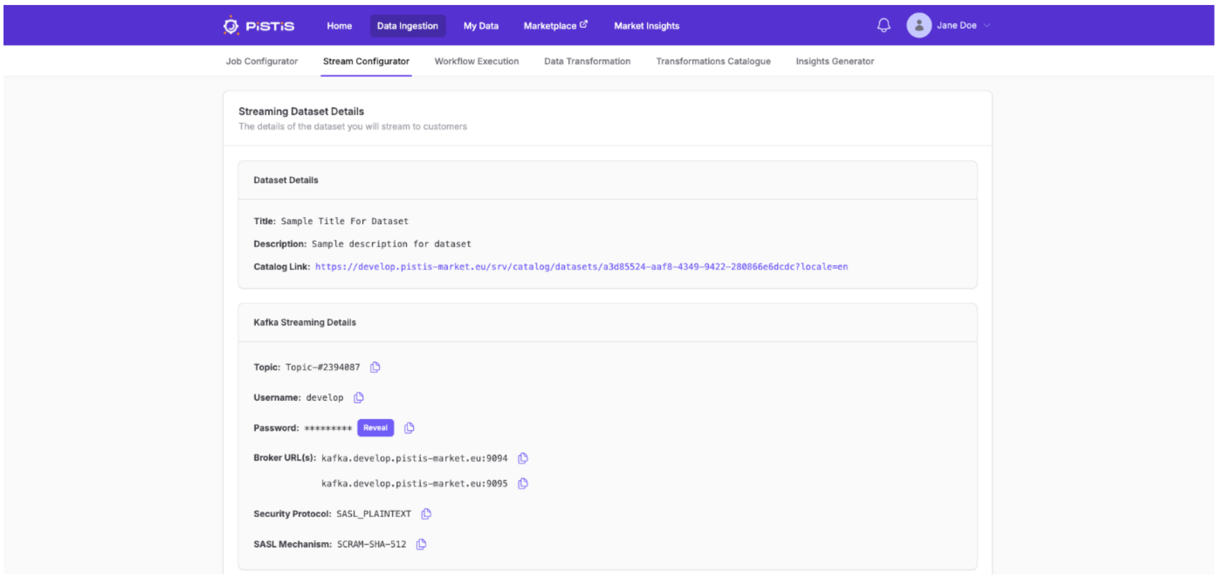
Streaming Dataset Details
The details of the dataset you will stream to customers

Dataset Title

Dataset Description

Figure 8: Registering a Kafka Stream – Inserting Stream Descriptions

Following this a Kafka topic is created for the specific Dataset and the system provides to the user (Data Provider) the topic name, the broker addresses and the credentials he/she should use so that he/she can send over the stream to the Kafka hosted in his/her Data Factory (as shown in the following figure). Once this is done, the Data Factory Kafka is collecting the streamed data.



Streaming Dataset Details
The details of the dataset you will stream to customers

Dataset Details

Title: `Sample Title For Dataset`

Description: `Sample description for dataset`

Catalog Link: `https://develop.pistis-market.eu/srv/catalog/datasets/a3d85524-aaf8-4349-9422-280866e6dcdc?locale=en`

Kafka Streaming Details

Topic: `Topic-#2394087`

Username: `develop`

Password: `*********`

Broker URL(s): `kafka.develop.pistis-market.eu:9094`
`kafka.develop.pistis-market.eu:9095`

Security Protocol: `SASL_PLAINTEXT`

SASL Mechanism: `SCRAM-SHA-512`

Figure 9: Registering a Kafka Stream – Stream Details

2.1.2.9 Developer Documentation

The developer documentation for the Streaming Data Check-In component may be accessed on the official PISTIS project documentation site at the following link: https://docs.pistis-market.eu/developers/streaming-data/introduction

2.1.2.10 Source Code

The source code developed for the Streaming Data Check-In is maintained in GitHub under the PISTIS project and can be accessed via the respective link: https://github.com/PISTIS-Platform/components-monorepo for the backend and, https://github.com/PISTIS-Platform/platform-frontend/ for the Data Factory UI.

2.2 Data Transformation

2.2.1 Component Description

Data transformation component aims at providing the possibility of performing some preprocessing tasks on the datasets to be handled by the PISTIS platform. These transformations can be very useful in order to improve the quality of the dataset, handling some aspects of the dataset that are commonly considered to diminish its value (e.g. missing values, wrong values, unformatted strings, etc.).

In order to perform that dataset preprocessing, a set of transformations can be defined in order to be applied over the dataset, setting up the transformation to apply, some detailed settings depending on the given transformation to apply, some filtering on the fields or registries to be transformed, etc.

2.2.2 Main Improvements in Beta Version

The following improvements have been introduced in beta version:

- Data transformation implementation template has been refactored.
- CSV file separator identification has been improved.
- A new version of the GUI has been implemented with the following features:
    - Improved UX: allowing the definition of transformations with no need to write any configuration file, just adding different transformations from their own GUI cards.
    - Fully automatic card generation approach: The GUI gets the current version of the transformation catalogue and automatically generates cards for each transformation available, including the input parameters defined for each transformation.
    - Improved composite transformation JSON generator: The GUI allows the addition (and removal) of different transformations to a transformation configuration JSON and its visualization in real time.

    - Transformation testing is still available: allowing the execution of transformations on a given dataset for testing purposes
- New transformations have been implemented:
    - Data Normalization
    - Column removal
    - Outlier Management

2.2.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data consumer	Get a list of available transformations to perform on PISTIS platform	I can choose what transformation to use	Alpha	Get a list with all the transformations offered by the PISTSI platform	Done	PISTIS.OUS.03
UC_02	Data consumer	Perform a PISTIS offered transformation on a dataset using the PISTIS platform	I can get my dataset modified	Alpha	The dataset has been updated following the transformation requested	Done	PISTIS.OUS.03
UC_03	Data Provider	Manage Data Transformations from GUI	Have data available in Pistis ecosystem	v1.00	Data stored properly in Factory Data Storage after applying transformations defined by the user.	Upcoming	PISTIS.OUS.03
UC_04	Data consumer	Get an improved set of transformations available	I can apply a wider set of transformations to my dataset	Beta	The catalogue of available transformations in beta version is bigger than the catalogue in alpha version	Done	PISTIS.OUS.03

2.2.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	PISTIS provides a definition of the data transformations supported	US_01, US_03	The component should provide a formal definition of the data transformations supported along with the format the request should be formatted
FR_02	PISTIS allows the transformation of a given dataset according to the definition of the transformation requested	US_02, US_03	

2.2.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Security	NFR1	PISTIS ensures that only authorised user can transform datasets.
Compatibility	NFR2	The component must be able to accept the input data in the CSV format (Pandas DataFrame compliant).
Compatibility	NFR3	Component transformation definition input must be compliant with the json schema of the transformations supported.

2.2.6 Component Architecture

As it can be seen in the following diagram, the component consists of two main logic modules: the first module is responsible for checking the validity of the transformations defined in the API call to be processed and the second module is responsible for the application of the logic of the transformations. The second module is the one responsible for loading the dataset passed as input and perform the transformation requested.

To that end, the component transformations have been designed in a modular and isolated way, defining a transformation template that ease the development of new transformations (just defining the schema of its required parameters and the logic of the transformation itself). Each one of these implemented transformations are scanned on deployment time and added automatically to the component transformation catalogue offered via GET API call.

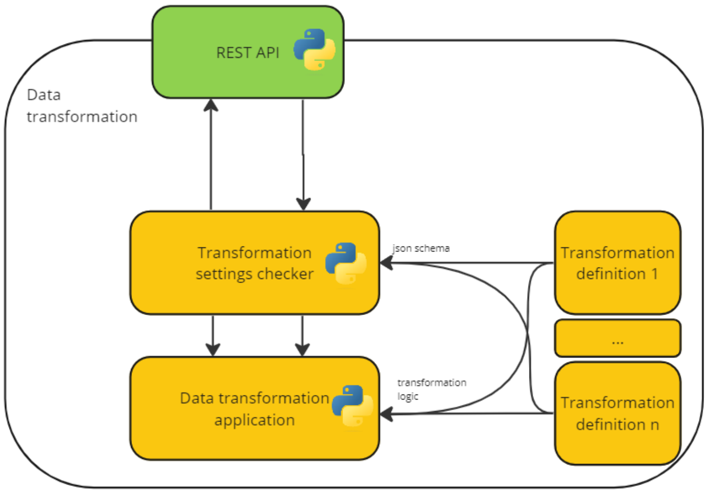

REST API
Data transformation
Transformation settings checker
json schema
Transformation definition 1
...
Data transformation application
transformation logic
Transformation definition n

Figure 10: Data Transformation Component's Internal Architecture

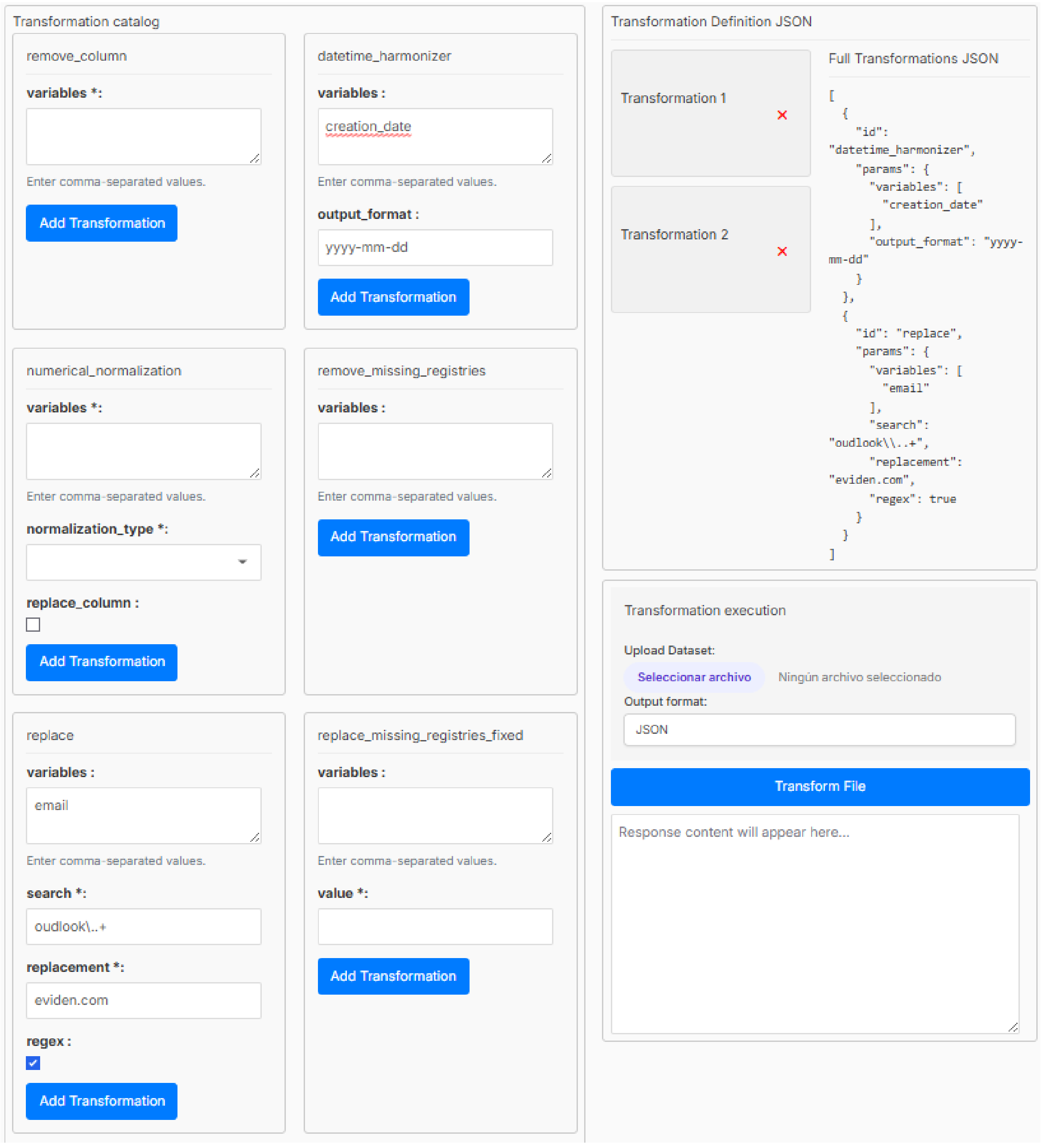
2.2.7 Technology Background

Data transformation will be based on Python, exploiting commonly used frameworks for data handling such as pandas, which ease the loading and processing of datasets by means of optimized data structures such as DataFrames (an optimized python-based data structure to handle datasets). Current version supports available data transformations listing (including:

string replacement, missing values replacement using fixed or statistical values or missing values removal) and its application.

2.2.8 GRAPHICAL USER INTERFACE

The UI of this component highly depends on the transformations defined. In case a dynamic GUI can be implemented, this GUI should consume the response from the GET API call in order to know which transformations are defined in the component and the different inputs that each transformation will require. In beta version of the GUI, this is already supported. Current version of the GUI offers, on the left side of the screen the data transformation catalogue displayed as cards, each one with its own title and input parameters. On the right side of the screen we can see a real-time JSON representation of the designed transformation on the top and the logic required to apply the designed transformation to a local file at the bottom. These three sections can be seen in the following screenshot:



Transformation catalog

remove_column

variables *:

Enter comma-separated values.

Add Transformation

datetime_harmonizer

variables : creation_date

Enter comma-separated values.

output_format : yyyy-mm-dd

Add Transformation

numerical_normalization

variables *:

Enter comma-separated values.

normalization_type *:

replace_column : ☐

Add Transformation

remove_missing_registries

variables :

Enter comma-separated values.

Add Transformation

replace

variables : email

Enter comma-separated values.

search *: oudlook\..+

replacement *: eviden.com

regex : ✓

Add Transformation

replace_missing_registries_fixed

variables :

Enter comma-separated values.

value *:

Add Transformation

Transformation Definition JSON

Transformation 1 ×

Transformation 2 ×

Full Transformations JSON

```
[
  {
    "id":
"datetime_harmonizer",
    "params": {
      "variables": [
        "creation_date"
      ],
      "output_format": "yyyy-
mm-dd"
    }
  },
  {
    "id": "replace",
    "params": {
      "variables": [
        "email"
      ],
      "search":
"oudlook\\..+",
      "replacement":
"eviden.com",
      "regex": true
    }
  }
]
```

Transformation execution

Upload Dataset: Seleccionar archivo Ningún archivo seleccionado

Output format: JSON

Transform File

Response content will appear here...

Figure 11: Data Transformation Component UI in beta version

2.2.9 Developer Documentation

Updated documentation for this component can be found at: https://docs.pistis-market.eu/developers/data-transformation/data-transformation

2.2.10 Source Code

The source code developed for the Data Transformation component is currently stored under an internal repository in ATOS premises until the license to be applied to it is decided.

The component's UI (frontend) is part of the platform-frontend repository and its code is maintained in GitHub under the PISTIS project and can be accessed via the respective link: https://github.com/PISTIS-Platform/platform-frontend

2.3 JOB CONFIGURATOR

2.3.1 COMPONENT DESCRIPTION

The Job Configurator oversees defining templates to support data pipeline jobs, as well as orchestrating them through the development of complicated workflows.

Job Configurator provides a high-level format for defining workflow and jobs to avoid only supporting those formats accepted by the workflow orchestration tool, which is Apache Airflow.

2.3.2 MAIN IMPROVEMENTS IN BETA VERSION

The following improvements have been introduced in beta version:

- GDPR Compliance Checking integration: During Check-In, the Job Configurator (JC) calls the GDPR Checker API and provides a URI. After retrieving the dataset from the Data Storage using that URI, the GDPR Checker will do the compliance checks and deliver any warnings or suggestions.
- Workflows execution scheduling: The purpose is to enable the periodic execution of workflows, which will allow for the periodic injection of batches of data connected with the same dataset. To that aim, two new fields have been added to the JC GUI, enabling the user to choose a precise workflow run time as well as a periodicity, which is now limited to three pre-sets: hourly, daily, or monthly.
- Dataset Encryption integration: To specifically indicate that the dataset should be encrypted and kept encrypted in the factory data storage, the user can choose a flag at the JC GUI level.
- Semantic Enrichment integration: During the check-in, after establishing the checking parameters, the JC displays a tiny slice of data (for example, 10 rows) and the Semantic Enrichment UI appears. The user makes mappings and stores them as a configuration. This configuration is then made accessible to the JC, who will run it in the end as part of the entire flow.
- Policy Editor integration: The default policy is provided once the dataset has been injected.
- Factory Catalogue integration: Data and Metadata registration.
- Factory Data Storage integration: Datasets storage.
- Linage Tracker integration: Once the dataset has been altered for a specific job during workflow execution to maintain lineage tracker traceability, it will be updated to create a new data distribution.
- IAM: Integration with Keycloak enables security.
- A new version of the GUI has been implemented with the following features:
  - Drag and Drop approach to define workflows as composition of services.
  - New flag to select dataset encryption.
  - New field to select a specific workflow run time.
  - New combobox added with a set of pre-sets (hourly, daily and monthly) to setup periodic workflow executions.

- New graphical sections for each service chosen in the workflow to complete input parameters.

2.3.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data Provider	Define a data pipeline related Job	Support data check-in, data transformation and analytics insights tasks	Alpha	Job Template as Airflow DAG	Done	PISTIS.SOUS.01, PISTIS.SOUS.02, PISTIS.SOUS.03
UC_02	Data Provider	Define a workflow because of composing different jobs	Support data pipeline	Alpha	Workflow template as Airflow DAG	Done	PISTIS.SOUS.01, PISTIS.SOUS.02, PISTIS.SOUS.03
UC_03	Data Provider	Define and run a workflow from a GUI	Support data pipeline	Beta	DAG associated to workflow definition instantiated and executed properly over Apache Airflow.	Done	PISTIS.SOUS.01, PISTIS.SOUS.02, PISTIS.SOUS.03

2.3.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	Provide support to define and execute Jobs templates to support data pipeline related tasks.	UC_01	
FR_02	Provide support to orchestrate jobs to support data pipelines.	UC_02	

2.3.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Security	NFR1	PISTIS ensures that only authorised user can transform datasets.
Compatibility	NFR2	The component must be able to accept the input data in the following formats: CSV, TSV, XML, XLSX, JSON and Parquet.

2.3.6 Component Architecture

Figure 12 illustrates the internal architecture of the Job Configurator, which is primarily based on the use of Apache Airflow, namely on two directed acyclic graphs, or DAGs: a) Workflow DAG (WDAG) and b) Job DAG (JDAG).

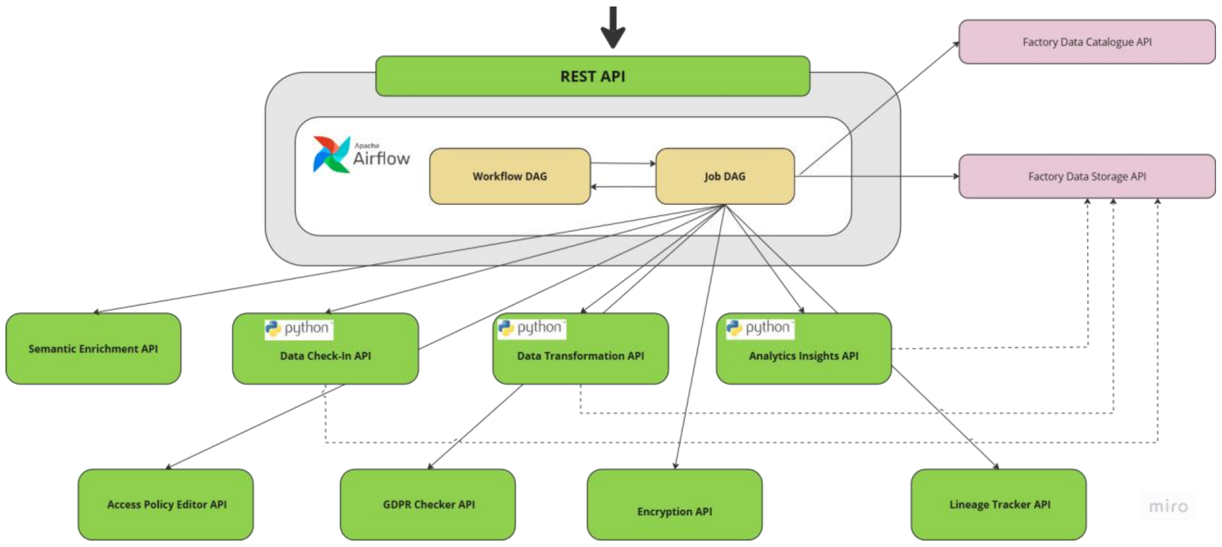


Figure 12: Job Configurator Architecture

The workflow execution is carried out via the WDAG, whose internal structure is depicted in Figure 13. Like the WDAG, the JDAG oversees carrying out a job or component in the context of PISTIS. Figure 13 illustrates the Job Dag's task-based organizational structure.

Specifically, the internal flow of the WDAG would be as follows:

1) Get the current Job from the workflow definition,
2) Resolve the mappings defined over the current job,
3) Trigger the Job DAG using the current job values,
4) Finally, if there are jobs pending, return to point 1), otherwise stop the workflow execution.

If we now focus on the Job DAG, its internal flow will be as follows:

1) Retrieve the data, using the required information from the data source.
2) Then, invoke the service endpoint specified in the job definition.
3) Check the storage policy and save the data response from point 2) in the Factory Data Storage or Memory as specified.
4) Return control to Workflow DAG.

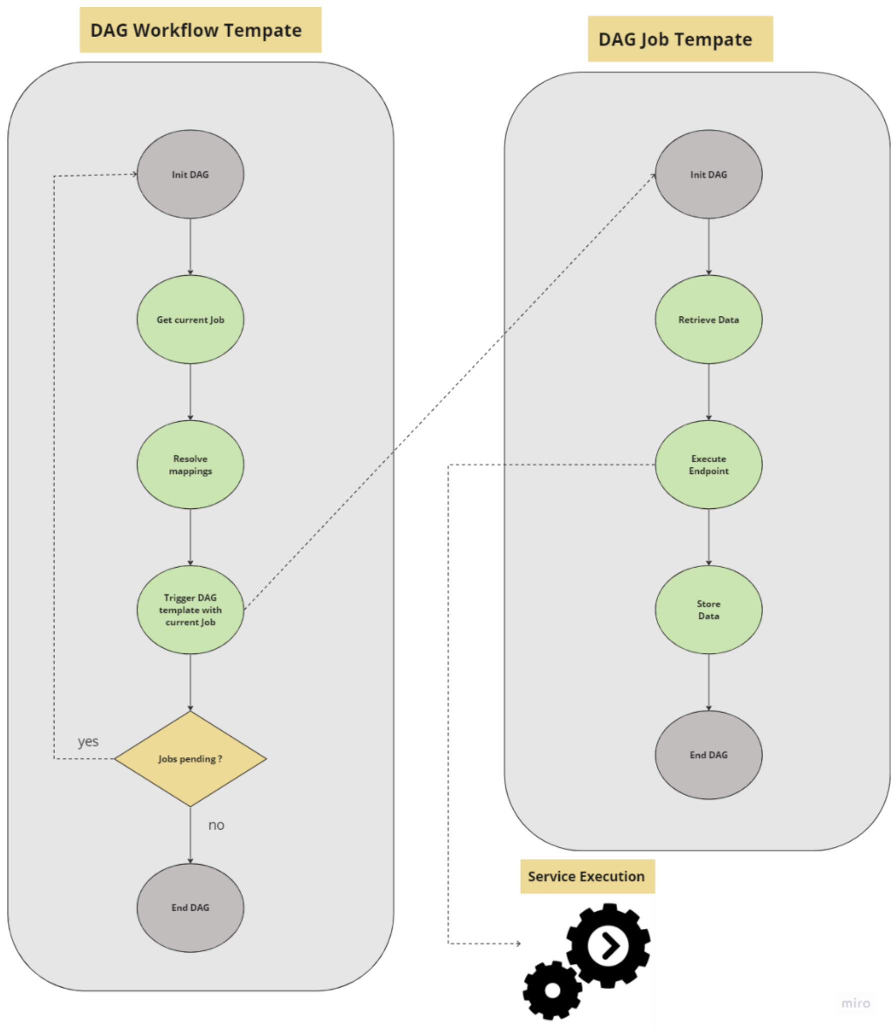


Figure 13: Workflow and Job DAGs

Finally, it is important to highlight that any service potentially orchestratable using the Job Configurator should satisfy the design principles shown in Figure 14.

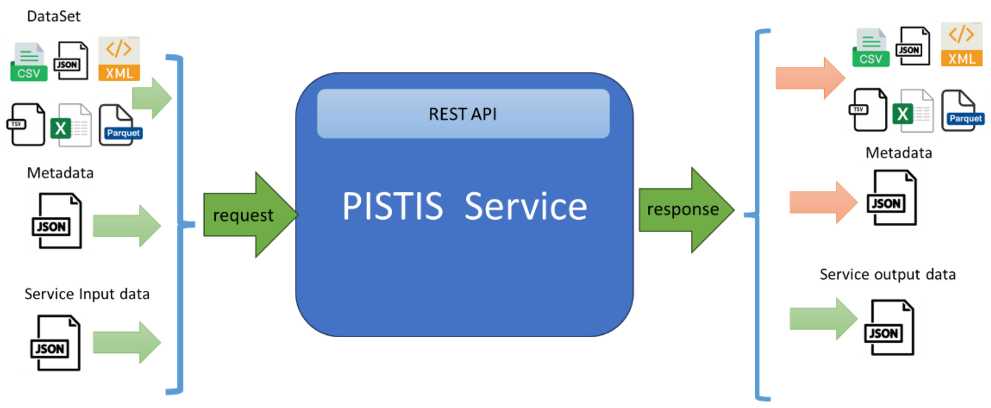


Figure 14: Service design principles approach 1

The chosen by the consortium format for working with data sources at the workflow level in the Alpha version of the component was CSV.

For the Beta version, the Job Configurator will be based on the selection and execution of services that support design principles described in Figure 14, extended the set of formats supported regarding version Alpha to: CSV, TSV, XML, XLSX, JSON and Parquet.

2.3.7 TECHNOLOGY BACKGROUND

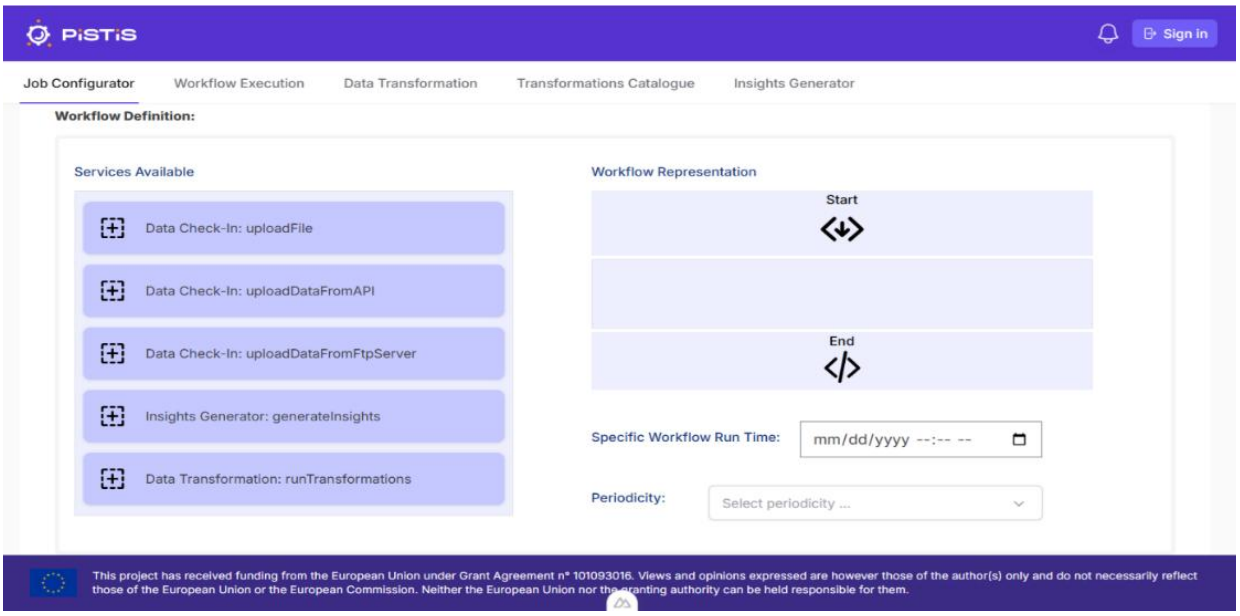
The main technology used for the Component is **Apache Airflow**.

2.3.8 GRAPHICAL USER INTERFACE

The UI definition for the Job Configurator has been included within the Data Pipeline UI bundle. The Job Configurator UI has been built on the idea of easily creating a workflow of jobs by dragging and dropping services and linking them to the next. Initially, the user begins with an empty panel named “Workflow Representation” onto which the user drags a series of services displayed just left in other panel named “Services Available”, as illustrated in Figure 15.

When a service is moved to the panel, either by selecting it or situating itself on it, a new panel at the bottom appears dynamically with the appropriate fields whose contents must be supplied for the service to run correctly. Figure 16 is an example of this using the data check-in service.

In addition to the composition and service instantiation parts, the UI will include a button for launching the workflow once it has been defined.



Workflow Definition:

Services Available

- Data Check-In: uploadFile
- Data Check-In: uploadDataFromAPI
- Data Check-In: uploadDataFromFtpServer
- Insights Generator: generateInsights
- Data Transformation: runTransformations

Workflow Representation

Specific Workflow Run Time: mm/dd/yyyy --:-- --

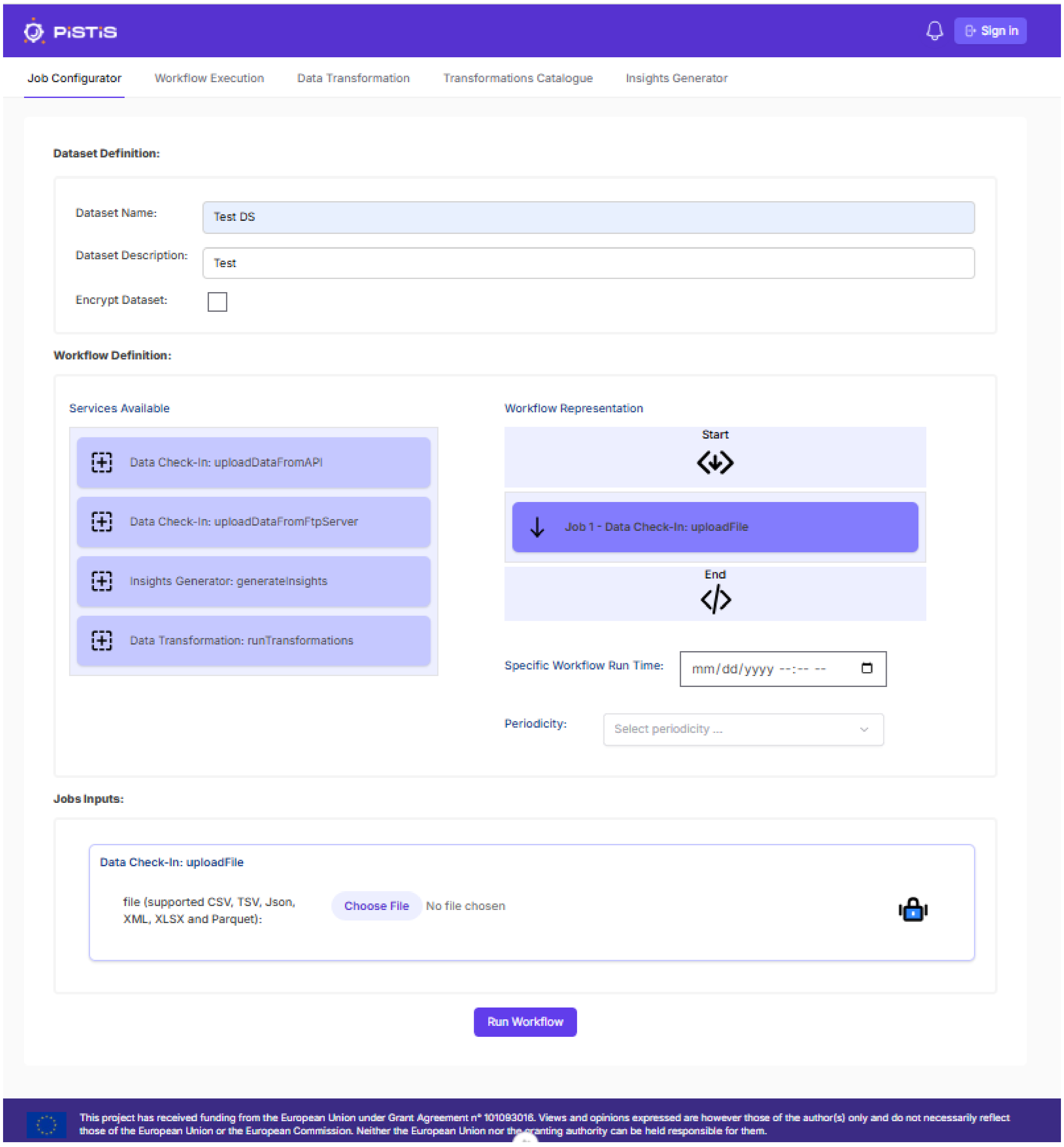
Periodicity: Select periodicity ...

This project has received funding from the European Union under Grant Agreement n° 101093016. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Figure 15: Job Configurator GUI Beta version.

When a service is moved to the panel, either by selecting it or situating itself on it, a new panel at the bottom appears dynamically with the appropriate fields whose contents must be supplied for the service to run correctly. Figure 16 Figure 16is an example of this using the data check-in service.

In addition to the composition and service instantiation parts, the UI will include a button for launching the workflow once it has been defined.



Dataset Definition:

Dataset Name: Test DS

Dataset Description: Test

Encrypt Dataset: ☐

Workflow Definition:

Services Available

- Data Check-In: uploadDataFromAPI
- Data Check-In: uploadDataFromFtpServer
- Insights Generator: generateInsights
- Data Transformation: runTransformations

Workflow Representation

Start

Job 1 - Data Check-In: uploadFile

End

Specific Workflow Run Time: mm/dd/yyyy --:-- --

Periodicity: Select periodicity ...

Jobs Inputs:

Data Check-In: uploadFile

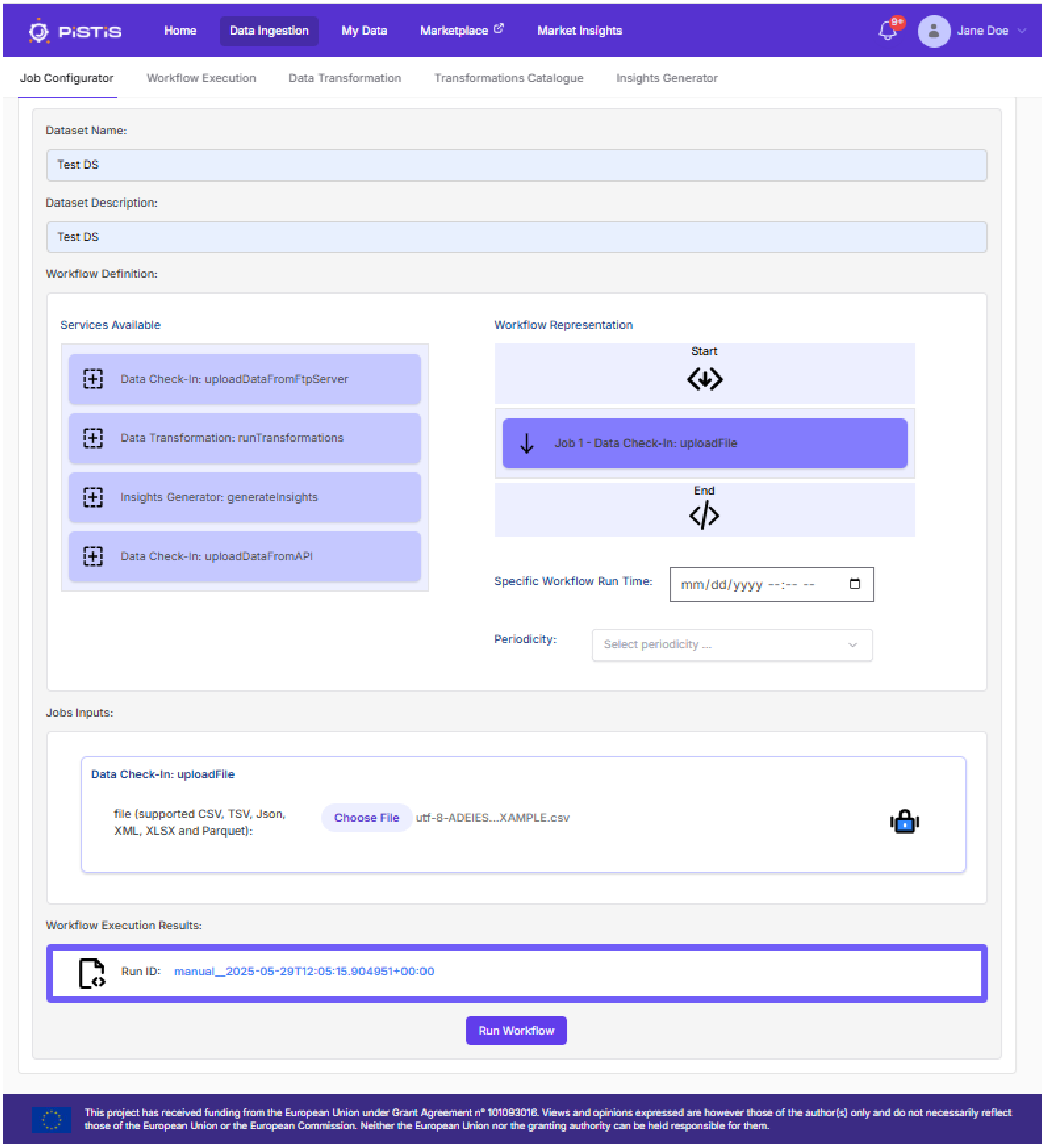
file (supported CSV, TSV, Json, XML, XLSX and Parquet): Choose File No file chosen

Run Workflow

This project has received funding from the European Union under Grant Agreement n° 101093016. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Figure 16: Data Check-In fields

Lastly, a run_id is supplied at the GUI's bottom to monitor the workflow's execution in real time and track its progress as shown in Figure 17.



Dataset Name: Test DS

Dataset Description: Test DS

Workflow Definition:

Services Available
- Data Check-In: uploadDataFromFtpServer
- Data Transformation: runTransformations
- Insights Generator: generateInsights
- Data Check-In: uploadDataFromAPI

Workflow Representation
- Start
- Job 1 - Data Check-In: uploadFile
- End

Specific Workflow Run Time: mm/dd/yyyy --:-- --

Periodicity: Select periodicity ...

Jobs Inputs:

Data Check-In: uploadFile

file (supported CSV, TSV, Json, XML, XLSX and Parquet): Choose File utf-8-ADEIES...XAMPLE.csv

Workflow Execution Results:

Run ID: manual__2025-05-29T12:05:15.904951+00:00

Run Workflow

This project has received funding from the European Union under Grant Agreement n° 101093016. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Figure 17: Run id (JC GUI)

Developer Documentation

The developer documentation for the Job Configurator component may be accessed on the official PISTIS project documentation site at the following link: https://docs.pistis-market.eu/developers/job-configurator/job-configurator

2.3.9 Source Code

The source code developed for the Job Configurator component is currently stored under an internal repository in ATOS premises until the license to be applied to it is decided.

The component's UI (frontend) is part of the platform-frontend repository and its code is maintained in GitHub under the PISTIS project and can be accessed via the respective link: https://github.com/PISTIS-Platform/platform-frontend

2.4 Analytics Engine

2.4.1 Component Description

In order to run ML/DL analytics pipelines and transform the primary data artefacts into insights, the Analytics Engine component can be automatically deployed to and self-hosted in computational resources that the user selects. This allows the trading of derivative data assets, such as the analysis's outputs. The engine could be integrated with the ML Model Registry to support a collection of pretrained AI models for various domains, which the PISTIS Data Consumers can utilize to expedite the creation of their AI pipelines.

Furthermore, the presence of an analytics engine will enable other modules of the overall PISTIS environment to benefit from its functionalities, such as enabling automatic data transformations, accommodating ML-based anonymisation activities, and running analyses relevant to the PISTIS market, such as trend identifications, predictions, and so on.

2.4.2 Main Improvements in Beta Version

The following improvements have been introduced in beta version:

- ML Recipes enabled to quickly develop high-quality models and deploy them to production.

2.4.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data Consumer	Generate an analytics playground custom deployment instruction	Deploy a playground instance	Alpha	Check deployment file	Done	PISTIS.OUS.03
UC_02	Data Consumer	Run some analytics	Run analytics	Beta	Enabling playground ecosystem	Done	PISTIS.OUS.03
UC_03	Data Consumer	Train an analytics model	Create a new AI Model	Beta	Check AI Model	Done	PISTIS.OUS.03
UC_04	Data Consumer	Visualize analytics results	Have a visual analysis of the results	V1.00	Access to graphs associated with the	Upcoming	PISTIS.OUS.03

					analytics results		

2.4.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	Manage the ML lifecycle	UC1, UC2, UC3, UC4	

2.4.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Security	NFR1	PISTIS ensures that only authorised user can analyse datasets.

2.4.6 Component Architecture

The analytics engine component's internal architecture is composed of an integration of different technologies, including PostgreSQL and MLFlow. By means of this, we can offer a set of functionalities that rely on the modules that are part of the final combination of technologies provided by the component. These modules include:

- Tracking module: Allows to keep track of the different metrics and objects defined by the end user on each experiment.
- The Artefacts Management module: Allows the storage of objects tracked on the experiments.
- Playground UI module: Offers a UI to interact with the component playground.
- Notebook module: Lets the end user to view and execute Jupyter notebooks.
- ML Pipeline Engine module: Is the module responsible for the execution of ML pipelines.
- ML Pipeline Definition module: Allows the definition of ML pipelines (in this case named as MLFlow recipes)

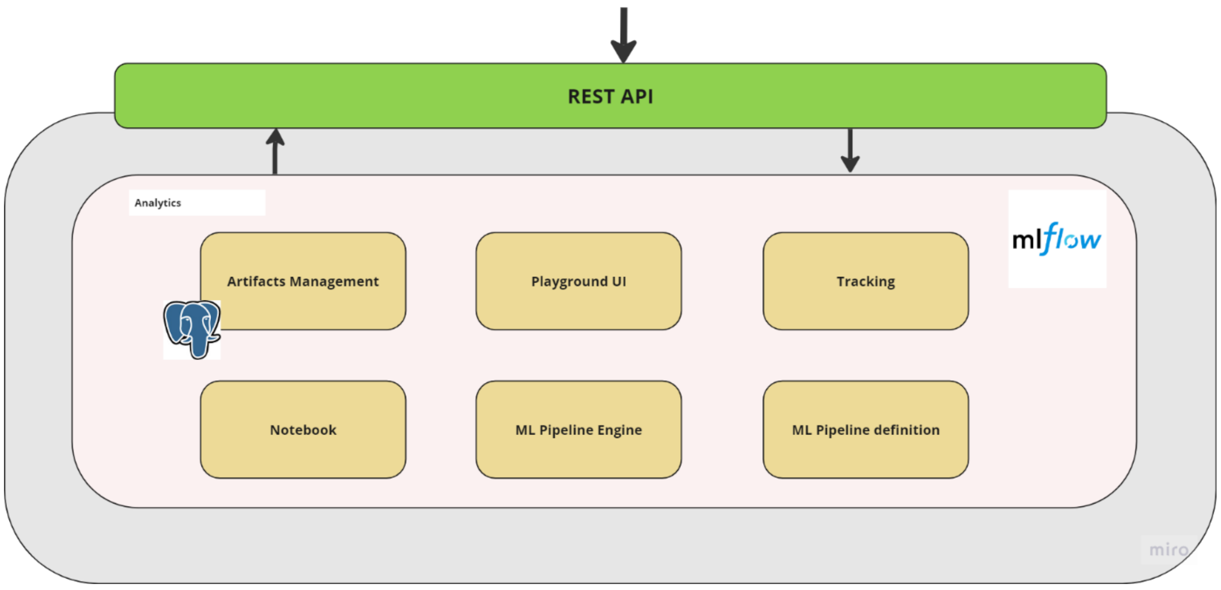


Figure 18: Analytics Engine Architecture

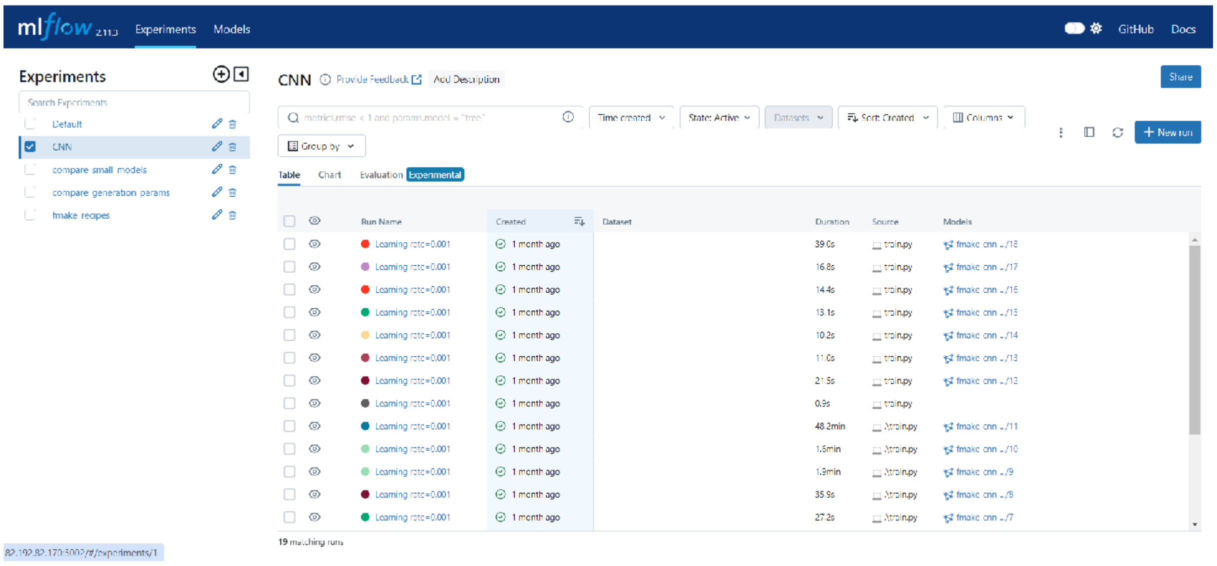
2.4.7 Technology Background

The main technology used for the Component is **MLFLow** and **PostgreSQL**.

2.4.8 Graphical User Interface

This component is powered primarily by MLFlow, which provides a proper GUI for managing the different experiments created in order to perform different analytics on the data, by means of tracking both artefacts and metrics resultant from these experiments. This tracking allows the creation and management of those experiments as well as the visualization of the different executions carried out in them, showing all the relevant values identified (and tracked) by the end user. This allows to see the evolution of the different outcomes of an experiment in accordance with the different inputs generated, offering the end user access to all the tracked artefacts that lead to that output.

Figure 19 shows the main screen of the component for experiment management, where the different experiments generated will be listed out.



Run Name	Created	Dataset	Duration	Source	Models
🟥 Learning rate=0.001	1 month ago		39.0s	train.py	fmake cnn .../18
🟪 Learning rate=0.001	1 month ago		16.8s	train.py	fmake cnn .../17
🟥 Learning rate=0.001	1 month ago		14.4s	train.py	fmake cnn .../16
🟩 Learning rate=0.001	1 month ago		13.1s	train.py	fmake cnn .../15
🟨 Learning rate=0.001	1 month ago		10.2s	train.py	fmake cnn .../14
🟥 Learning rate=0.001	1 month ago		11.0s	train.py	fmake cnn .../13
🟥 Learning rate=0.001	1 month ago		21.5s	train.py	fmake cnn .../12
⬛ Learning rate=0.001	1 month ago		0.9s	train.py	
🟦 Learning rate=0.001	1 month ago		48.2min	.\train.py	fmake cnn .../11
🟩 Learning rate=0.001	1 month ago		1.6min	.\train.py	fmake cnn .../10
🟩 Learning rate=0.001	1 month ago		1.9min	.\train.py	fmake cnn .../9
🟥 Learning rate=0.001	1 month ago		35.9s	.\train.py	fmake cnn .../8
🟩 Learning rate=0.001	1 month ago		27.2s	.\train.py	fmake cnn .../7

19 matching runs

Figure 19: Analytics Engine GUI

2.4.9 Developer Documentation

Since MLFlow, a third-party technology, forms the basis of the Analytics Engine component, the component's documentation is publicly accessible at https://mlflow.org/docs/1.30.0/index.html.

2.4.10 Source Code

The Analytics Engine component is based on 3rd party technology called MLFlow whose open-source repository is accessible at https://github.com/jupyterlab/jupyterlab.

2.5 Data Enrichment

2.5.1 Component Description

The data enrichment component in PISTIS is part of the Data Ingestion and Transformation module located within an organization. This component, like the data transformation component, enhances available data assets to increase their trading value. Data enrichment typically involves harmonizing data using additional sources, combining various formats and naming conventions into a standardized format for analysis. The main goal of this module in PISTIS is to convert a raw dataset file into a structured dataset that aligns with specific domain data models. This process allows users to upload raw datasets as files, transforming them into PISTIS datasets for availability in the Marketplace. The module supports the parsing and transformation of CSV, TXT, JSON, and XLS files.

It consists of a frontend and backend: the backend handles file parsing and provides access to PISTIS Data Model properties, while the frontend offers an interface for users to view datasets and select suitable properties. Once a user selects a dataset file for enrichment and identifies the appropriate properties for each column, the file is transformed into a SQL table with a new schema, which is then inserted into the Factory Data Storage and registered as a new distribution in the Factory Data Catalogue.

This new distribution includes the dataset ID from the Factory Data Storage and basic metadata, along with the newly created table schema. This schema can be reused for enriching additional raw dataset files, allowing for automated execution of the enrichment process and reducing the need for manual mapping. This feature is especially beneficial for data streams, as it enables users to perform enrichment automatically to all incoming data streams if an initial version is manually mapped and the schema is stored in the Factory Data Catalogue. Users can also convert a file into multiple tables with different schemas and query the enriched datasets from the Factory Data Storage.

2.5.2 Main Improvements in Beta Version

For the beta release version of this component, several changes and improvements have been added both to the GUI and to the backend. This version is more user-friendly, scalable and supports automatic execution of the data model mapping process. Additional features available in the beta release version are:

UI Improvements

- The page for users to confirm or add a dataset header has been updated with a new button for this process.
- Datasets are now displayed in a more compact view.
- Users can now assign a name to the distribution created in the Factory Data Catalogue at the end of the process.
- The properties of the PISTIS Data Model in the drop-down menu now include an "**i**" information button that links to the property's URI, providing users with additional information about the property, including its domain and range.
- A "**Reset all**" button has been added to reset all column name values, allowing users to restart the mapping process if desired.
  More informative error messages have been introduced to help users understand the reasons for failures.

Backend Improvements

- A new endpoint has been introduced to support the automatic execution of the enrichment process, which takes a new raw dataset and a dataset ID from the Factory Data Catalogue to retrieve the schema for enrichment.
- The schema stored in the Factory Data Catalogue will now include the URI and names of all properties.
- Error handling has been improved for better reliability.

2.5.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data provider	Fetch a file I uploaded and apply a transformation	A specific transformation can be performed	Alpha	Has access to the files	Done	PISTIS.OUS.01
UC_02	Data provider	Fetch the data models stored in PISTIS models repository	Keep an updated list of data models to show to the user	Alpha	Has access to the data models	Done	PISTIS.OUS.06
UC_03	Data provider	Transform my datasets to a data model	The datasets follow a domain specific standard	Alpha	Has access to the files	Done	PISTIS.OUS.06
UC_04	Data provider	Store the transformed dataset in the Factory Data Storage	The transformed dataset is available in the PISTIS platform	Alpha	The format is supported by the Factory Data Storage	Done	PISTIS.OUS.01
UC_05	Data provider	Automatically perform enrichment for a new dataset using a schema stored in the Catalogue	Manual mapping to a data model does not need to be performed	Beta	Schema is stored in the catalogue	Done	PISTIS.OUS.01

2.5.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	The Data Enrichment component can access a file that the user has uploaded.	US_01	
FR_02	The Data Enrichment component gives the user access to domain specific data models to transform their dataset to.	US_02	
FR_03	The Data Enrichment component allows the user to assign a data model present in the PISTIS Data Models repository to a dataset.	US_03	
FR_04	The Data Enrichment component allows the user to save the transformed dataset into the Factory Data Storage.	US_04	

2.5.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Performance efficiency	NFR1	The Data Enrichment component will allow the user to fetch files, assign a data model and save the new dataset without causing any delays.
Compatibility	NFR2	The Data Enrichment component can be integrated with the Factory Data Storage, Factory Data Catalogue and PISTIS Data Models Repository.
Reliability	NFR3	The Data Enrichment component will allow user the access to the latest available data models in the PISTIS Data Models Repository.
Reliability	NFR4	All endpoints of the Data Enrichment component are always functional and proper error messages are provided when a request fails.
Security	NFR5	The Data Enrichment component will be secured with the Identity and Authorization Manager and Access Policies Manager.
Portability	NFR6	The Data Enrichment component is containerized and can be deployed in hardware that supports Docker.

2.5.6 Component Architecture

This section describes the component architecture of the Data Enrichment service. The figure below illustrates the various modules within this service and their interactions. The GUI guides users through selecting a dataset and an appropriate data model. The backend is developed in Python using Flask to provide API endpoints, while SQLite is utilized to cache the dataset as the user makes their selection in the frontend. In PISTIS, this service operates through integration with the PISTIS Models Repository, Factory Data Catalogue, and Factory Data Storage. A Swagger UI is added to the backend to provide detailed API descriptions.

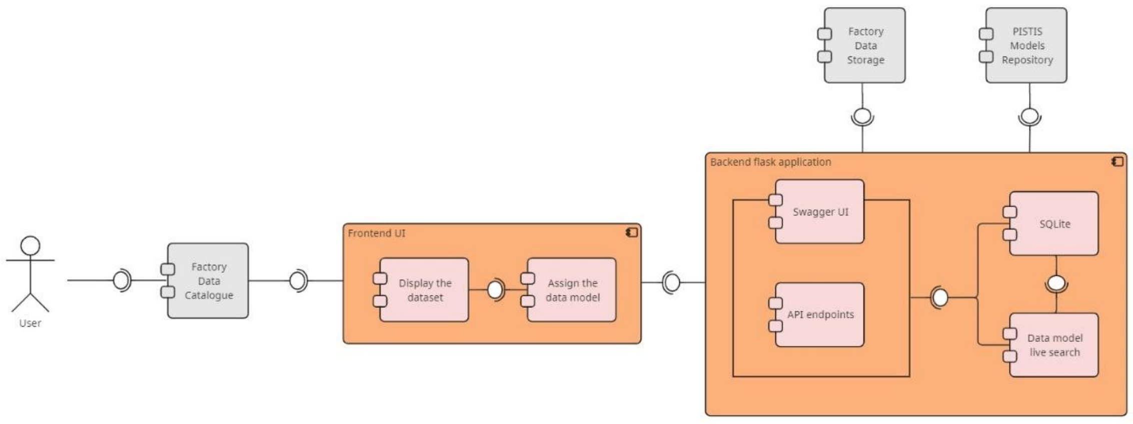


Figure 20: Data Enrichment Internal Architecture

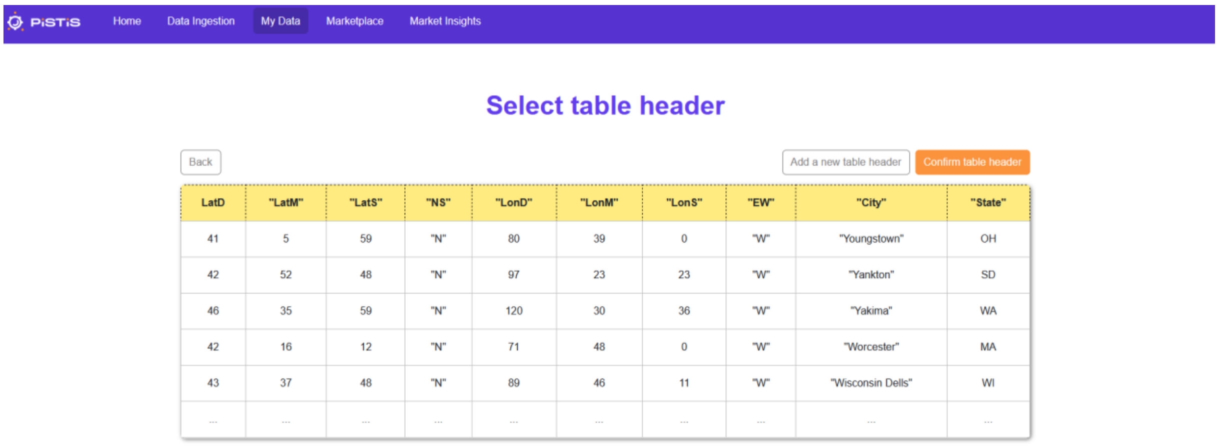
2.5.7 TECHNOLOGY BACKGROUND

The backend of the data enrichment module consists of a Python Flask App, and an SQLite database. The Flask App serves a RESTful API that communicates with the frontend of this component.

The frontend of the data enrichment module is built in Vue.js 3.0., using Pinia as the state management framework and bootstrap CSS library.

2.5.8 GRAPHICAL USER INTERFACE

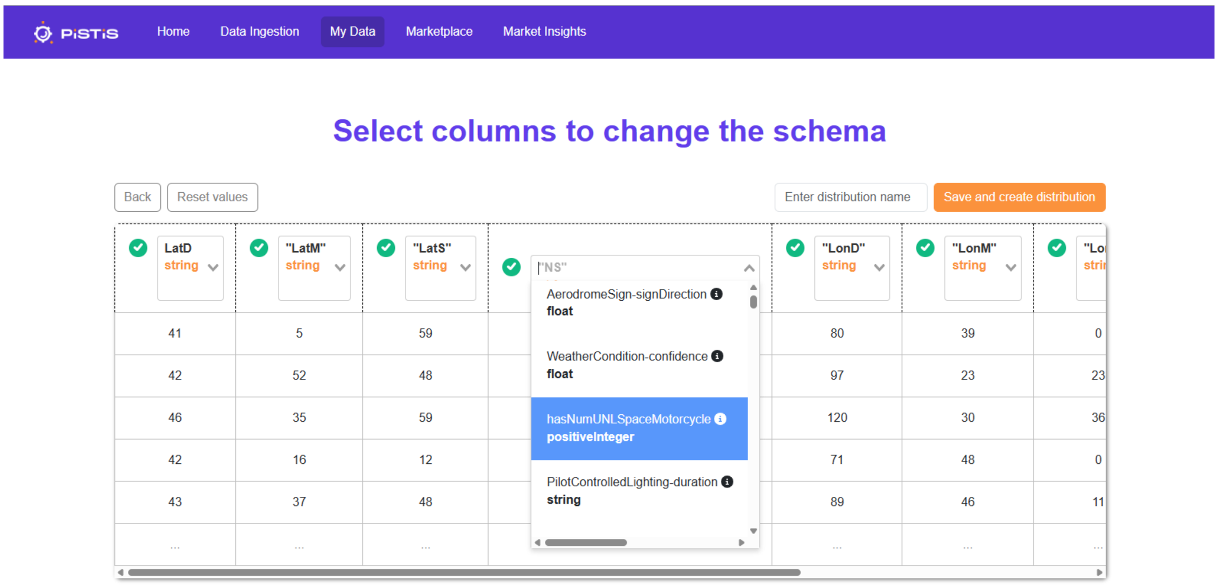
The first page of the Data Enrichment interface prompts the user to confirm the table header that will be enriched with properties from the PISTIS data model. After confirming the header, the user proceeds to the next page, where they can select specific properties for each column. If no table header is available, the user has the option to add a new one.



Select table header

LatD	"LatM"	"LatS"	"NS"	"LonD"	"LonM"	"LonS"	"EW"	"City"	"State"
41	5	59	"N"	80	39	0	"W"	"Youngstown"	OH
42	52	48	"N"	97	23	23	"W"	"Yankton"	SD
46	35	59	"N"	120	30	36	"W"	"Yakima"	WA
42	16	12	"N"	71	48	0	"W"	"Worcester"	MA
43	37	48	"N"	89	46	11	"W"	"Wisconsin Dells"	WI
...	...	...	...	...	...	...	...	...	...

Figure 21: Display the dataset



Select columns to change the schema

LatD string	"LatM" string	"LatS" string	"NS"	"LonD" string	"LonM" string	"Lo stri
41	5	59		80	39	0
42	52	48		97	23	23
46	35	59		120	30	36
42	16	12		71	48	0
43	37	48		89	46	11
...	...	...		...	...	...

AerodromeSign-signDirection
float

WeatherCondition-confidence
float

hasNumUNLSpaceMotorcycle
positiveInteger

PilotControlledLighting-duration
string

Figure 22: Select the properties of the data model

After confirming the table header, selecting a column reveals a dropdown menu listing all properties from the PISTIS data model. Each property includes an information icon that links to its corresponding ontology URL. This selection process can be repeated for every column. If needed, the user can restart the process using the "Reset values" button. Once all selections are made and a distribution name is provided, clicking the "Save and create distribution" button stores the dataset in the Factory Data Storage and adds a new distribution entry referencing it in the Factory Data Catalogue.

2.5.9 Developer Documentation

Data Enrichment service is accessible from the distributions in the Factory Data Catalogue UI. The developer documentation of this service is available at: https://docs.pistis-market.eu/developers/factory-data-storage/intro.

2.5.10 Source Code

Data Enrichment service consists of a frontend module built in Vue.js and a backend module built using Python Flask. The source code of these modules is accessible here:

1. Data Enrichment Frontend: https://github.com/PISTIS-Platform/data-enrichment-frontend
2. Data Enrichment Backend: https://github.com/PISTIS-Platform/data-enrichment-backend

2.6 Data Quality Assessment

2.6.1 Component Description

The Data & Metadata Quality Assessment component ensures the quality and consistency of data and metadata within the PISTIS system.

It provides two main functionalities:

- **Metadata Assessment:** This module checks and validates metadata against the predefined Metadata model and returns the validation result together with a score of the result. It identifies missing metadata, validates data types and formats, and ensures adherence to data standards.
- **Data Assessment:** This module checks and validates structured data against quality rules extracted from distribution metadata and returns the validation results. It checks for data consistency, adherence to data quality rules, and identifies potential errors or anomalies. The validation process ensures that data is reliable, accurate, and usable for downstream applications. For this purpose, it uses the great expectations python library.

The Data & Metadata Quality Assessment component provides APIs for both metadata and data validation, allowing integration with various data management and processing tools. It also supports on-demand and scheduled validation runs, enabling proactive data quality monitoring.

2.6.2 Main Improvements in Beta Version

The main improvements to the DQA beta version include a restructure of the content data quality assessment backend workflow. This was a result of internal discussions around ensuring trust and transparency within the PISTIS platform. Specifically, whether data owners should be allowed to freely define data quality rules on their owned datasets. It was agreed that given the difference between data quality in a production pipeline and data quality in a marketplace, it was necessary to run quality assessments automatically, without customization by the data owner.

The main changes to the workflow revolve around, using the insight report generated by the Insight Generator component as the backbone of the DQA, which can then be supplemented by rules parsed from the feature to concept mapping defined during Data Enrichment.

Additionally, both the MQA and DQA now use the same RDF report storage and are visualized in using the same UI.

2.6.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data Consumer	Check the Quality of a particular Data	I can be sure before buying a data	Beta	Generate QA report as DQV for data	Done	PISTIS. OUS.0 4
UC_02	Data Consumer	Check the Quality of a particular Metadata with a general schema	I can be sure before buying a data	Alpha	Generate QA report as DQV for metadata	Done	PISTIS. OUS.0 4
UC_03	Data Provider	Automatically check the quality of the Data I provide	I know the quality of my data and can get the expected value	Beta	Generate QA report as DQV for data	Done	PISTIS. OUS.0 4
UC_04	Data Provider	Automatically check the quality of the Metadata I provide	I can be sure Data Consumers can find my data	Alpha	Generate QA report as DQV for metadata	Done	PISTIS. OUS.0 4
UC_05	Data Provider	Represent the best possible quality of my data	I can achieve a higher valuation	Beta	Display QA report for data	Done	PISTIS. OUS.0 4
UC_06	Data Consumer	Check the Quality of a particular Metadata with a PISTIS metadata schema	I can be sure before buying a data	Beta	Generate QA report as DQV for metadata using PISTIS schema	Done	PISTIS. OUS.0 4
UC_07	Data Consumer	Request the assessment of user defined quality rules against a particular published dataset	I can be sure the data satisfies my use case requirements	Version 1.0	Generate consumer designed quality assessment requests	Upco ming	PISTIS. OUS.0 4

2.6.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	The DQA should provide the ability to automatically check the Quality of the Metadata.	US_04, US_06	
FR_02	The DQA should provide the ability to check the Quality of the Data after the user defined the structure of it.	US_05, US_03	
FR_03	The DQA should provide the ability to make the assessment results available to interested users.	US_01, US_02	

2.6.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Functional Suitability	NFR1	The DQA provides with a data model to collect all information for data asset publication or exchange.
Performance efficiency	NFR2	The DQA fetch all information from the factory components to run a quality assessment.
Compatibility	NFR3	The DQA will easily integrate with all directly connected catalogues and components.
Usability	NFR4	The DQA should be automatically run when a dataset distribution has a content update.
Reliability	NFR5	All endpoints of the DQA are functioning and clear error messages are provided when a request fails.
Security	NFR6	The DQA can be created only as the result of a distribution content update by a registered Data Provider.
Portability	NFR7	The DQA is containerized and can be deployed in hardware that supports Docker.

2.6.6 Technology Background

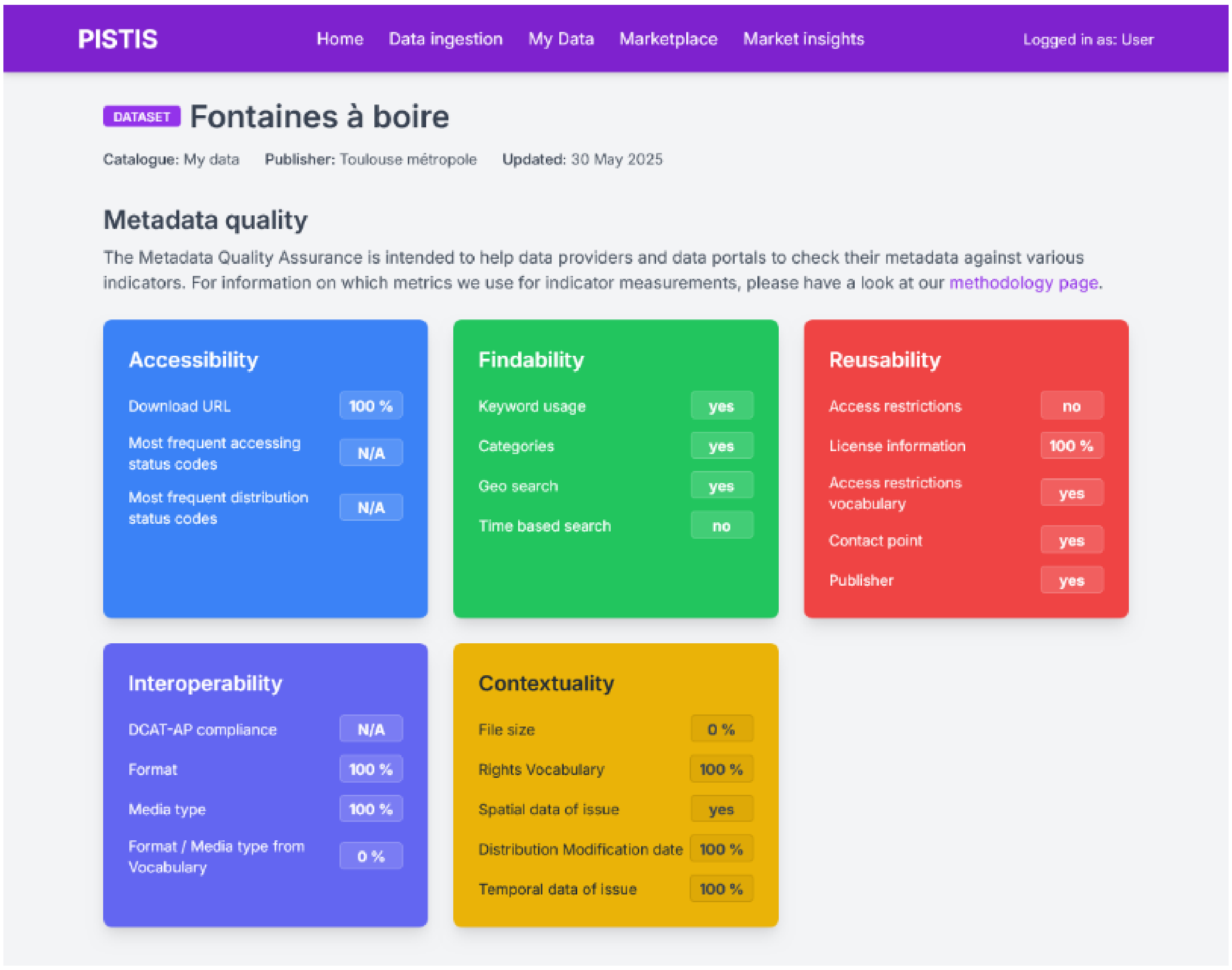
- The metadata assessment functionalities will be based on Fraunhofer FOKUS' *piveau metrics* and uses the *piveau pipe* for job management. The triple store of the data catalogue is used to store the result as linked data, a MongoDB is used for aggregating and caching the result to display it.
- The data content assessment uses Great Expectations as a framework to define and validate data expectations. It allows for the creation of data pipelines and automated tests to ensure data quality and integrity.

2.6.7 Graphical User Interface

The UI for the quality assessment results can be divided into three parts:

1. The result for the dataset metadata
2. The result for the metadata of each of the datasets distributions metadata
3. The result for the data itself

So, the following screenshot shows the result of a result for a single dataset metadata. This result data is being retrieved from the metrics aggregator component. In cases where a metrics is calculated over the distributions, a percentage of the number of valid distributions is shown.



Fontaines à boire

Catalogue: My data Publisher: Toulouse métropole Updated: 30 May 2025

Metadata quality

The Metadata Quality Assurance is intended to help data providers and data portals to check their metadata against various indicators. For information on which metrics we use for indicator measurements, please have a look at our methodology page.

Accessibility

Download URL	100 %
Most frequent accessing status codes	N/A
Most frequent distribution status codes	N/A

Findability

Keyword usage	yes
Categories	yes
Geo search	yes
Time based search	no

Reusability

Access restrictions	no
License information	100 %
Access restrictions vocabulary	yes
Contact point	yes
Publisher	yes

Interoperability

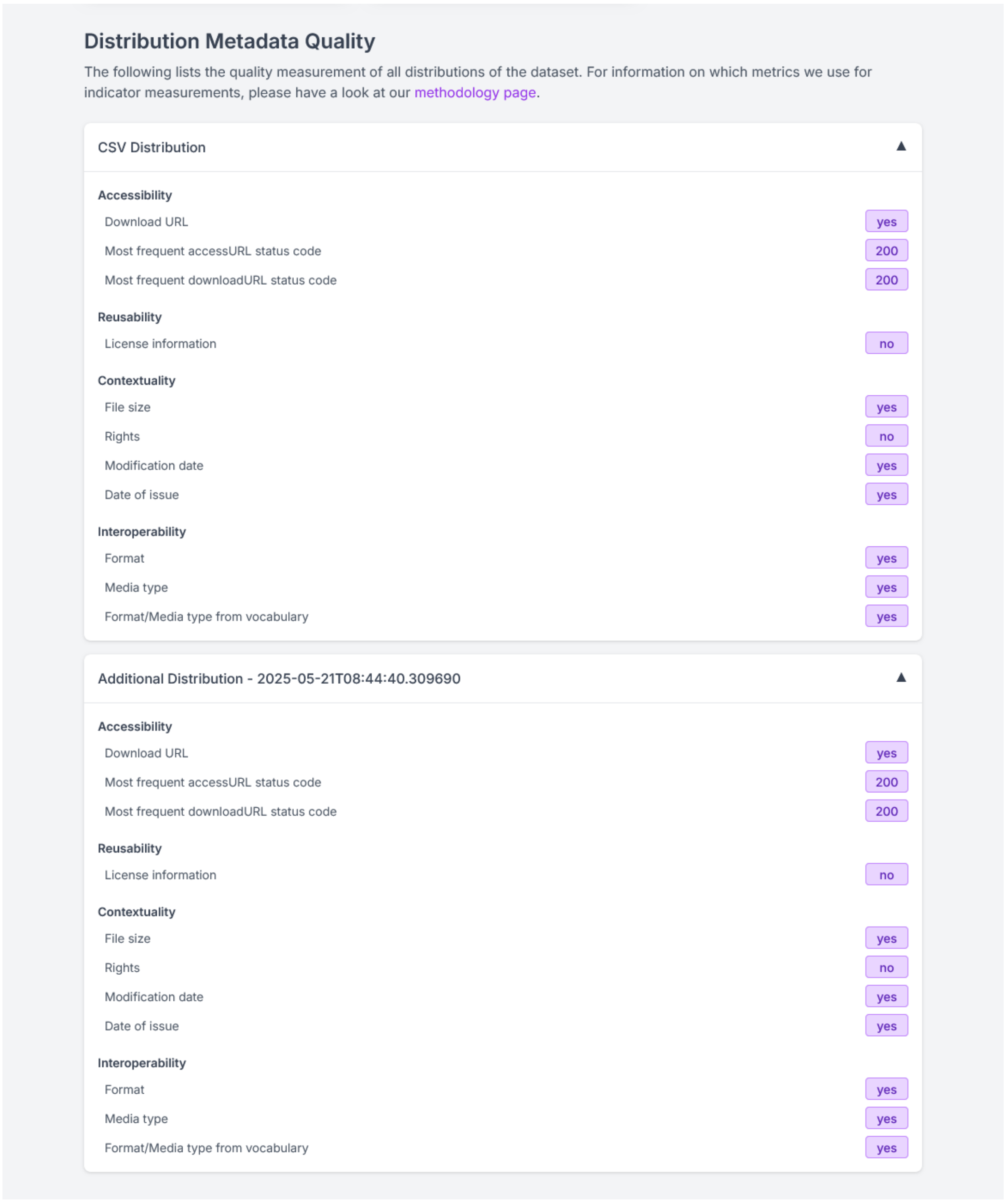
DCAT-AP compliance	N/A
Format	100 %
Media type	100 %
Format / Media type from Vocabulary	0 %

Contextuality

File size	0 %
Rights Vocabulary	100 %
Spatial data of issue	yes
Distribution Modification date	100 %
Temporal data of issue	100 %

Figure 23: Overview of the assessment result for a dataset

Then in the following figure the individual result for both of the datasets distributions is displayed. Each distribution metadata is validated on its own.



Distribution Metadata Quality

The following lists the quality measurement of all distributions of the dataset. For information on which metrics we use for indicator measurements, please have a look at our methodology page.

CSV Distribution ▲

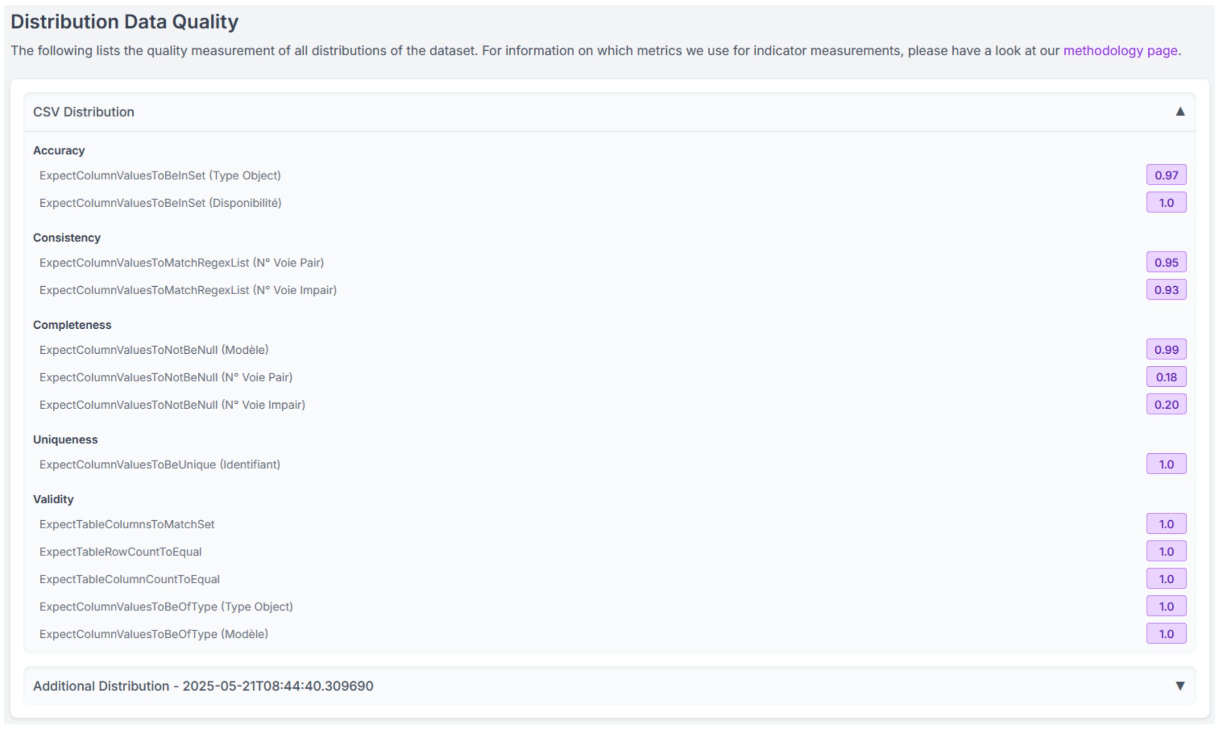
Accessibility	
Download URL	yes
Most frequent accessURL status code	200
Most frequent downloadURL status code	200
Reusability	
License information	no
Contextuality	
File size	yes
Rights	no
Modification date	yes
Date of issue	yes
Interoperability	
Format	yes
Media type	yes
Format/Media type from vocabulary	yes

Additional Distribution - 2025-05-21T08:44:40.309690 ▲

Accessibility	
Download URL	yes
Most frequent accessURL status code	200
Most frequent downloadURL status code	200
Reusability	
License information	no
Contextuality	
File size	yes
Rights	no
Modification date	yes
Date of issue	yes
Interoperability	
Format	yes
Media type	yes
Format/Media type from vocabulary	yes

Figure 24: The quality measurement results for a single distribution of this dataset

The following figure shows the visualization of a data quality assessment run on the content of a CSV distribution. Each data quality rule is evaluated individually and then grouped by its respective data quality dimension.



Distribution Data Quality

The following lists the quality measurement of all distributions of the dataset. For information on which metrics we use for indicator measurements, please have a look at our methodology page.

CSV Distribution ▲

Accuracy	
ExpectColumnValuesToBeInSet (Type Object)	0.97
ExpectColumnValuesToBeInSet (Disponibilité)	1.0
Consistency	
ExpectColumnValuesToMatchRegexList (N° Voie Pair)	0.95
ExpectColumnValuesToMatchRegexList (N° Voie Impair)	0.93
Completeness	
ExpectColumnValuesToNotBeNull (Modèle)	0.99
ExpectColumnValuesToNotBeNull (N° Voie Pair)	0.18
ExpectColumnValuesToNotBeNull (N° Voie Impair)	0.20
Uniqueness	
ExpectColumnValuesToBeUnique (Identifiant)	1.0
Validity	
ExpectTableColumnsToMatchSet	1.0
ExpectTableRowCountToEqual	1.0
ExpectTableColumnCountToEqual	1.0
ExpectColumnValuesToBeOfType (Type Object)	1.0
ExpectColumnValuesToBeOfType (Modèle)	1.0

Additional Distribution - 2025-05-21T08:44:40.309690 ▼

Figure 25: Example content quality assessment results

2.6.8 Component Architecture

2.6.8.1 *Metadata Quality Assessment*

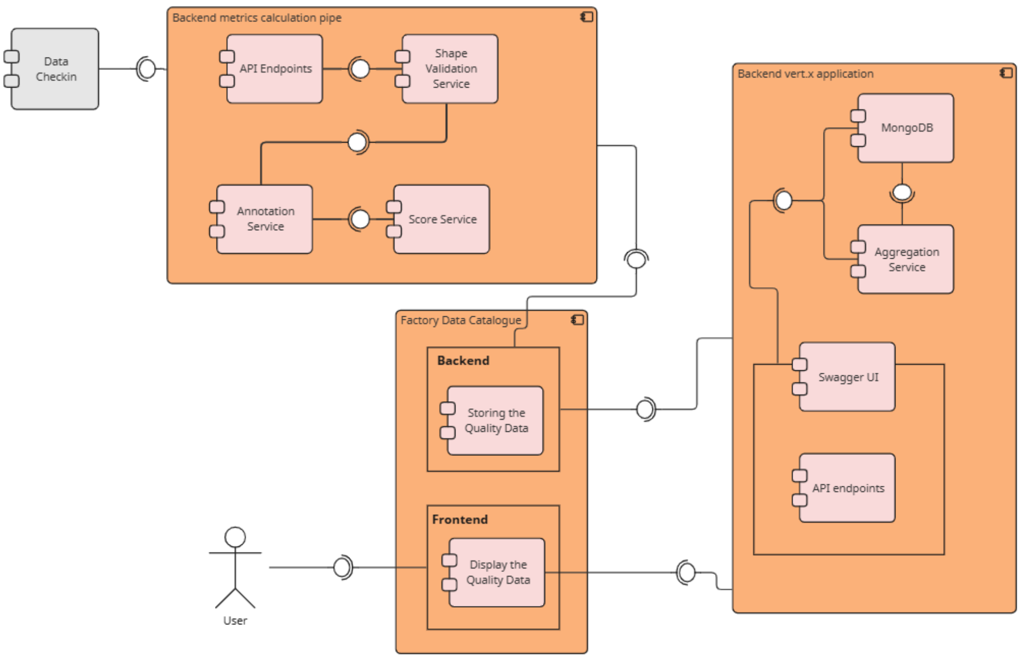


Figure 26: Metadata Quality Assessment service internal architecture

The MQA consists of three main layers, the pipeline layer, the services layer and the UI layer.

The pipeline layer is called during Data Check-In and consists of several microservices to assess the metadata quality and calculate the score. This result will then be stored as linked data in the data factories linked data storage.

The UI layer is integrated into the data catalogue, so that it is possible to see the quality of each dataset in the catalogue. To show this data it will talk to the services layer, which will provide a cached version of the result for a more performant access.

2.6.8.2 Data Quality assessment

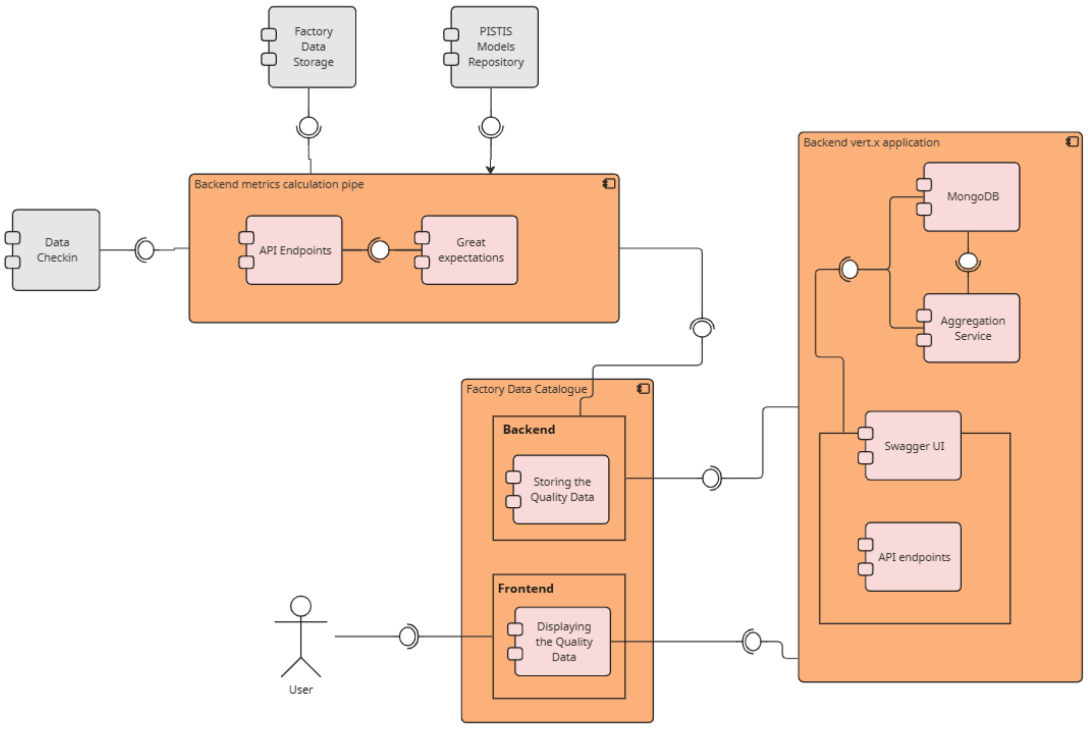


Figure 27: Data Quality Assessment service internal architecture

The DQA service will have a similar structure to the MQA. It will also have a pipeline layer to calculate the quality of the data after Data Check-In. This will utilize the open-source data quality platform great expectations.

The result will also be integrated into the data catalogue for easier access.

2.6.9 Developer Documentation

The Data Quality Assessment component documentation is available at the following URL: https://docs.pistis-market.eu/developers/data-quality-assessment/data-quality-assessment

2.6.10 Source Code

The Data Quality Assessment component source code is available in the following repository: https://github.com/PISTIS-Platform/data-quality-assessor.

The metadata quality assessment is based on an existing open source solution: piveau Metrics[5]. Several services from this project are used, integrated and configured to fit the needs for PISTIS. For the source code please refer to the upstream repositories as listed in the corresponding repository https://github.com/PISTIS-Platform/metadata-quality-assessment. The following services are used: the Metrics Annotator performs static analysis on the metadata, the Metrics Validating SHACL service checks a given RDF for conformity against a given SHACL schema, the Metrics Score calculates a numeric data quality rating depending on the analysis performed by the SHACL Validator and annotator, the Metrics Cache stores the quality measurements as documents in a MongoDB for fast retrieval and filtering and the Exporter stores the quality measurements as RDF in a Triplestore.

2.7 Data Insights Generator

2.7.1 Component Description

The Data Insights generator is a component that provides information about the structure and data types of a given dataset in order to ease the understanding of a dataset for the final user.

The component is expected to receive a given dataset and map it to a python pandas DataFrame. From that input dataset, a report on the different fields of the dataset is expected to be provided, including some information such as the data type of each field, number of missing elements, different values in categorial values, some statistical analytics on numerical data, etc.

2.7.2 Main Improvements in Beta Version

For this beta version, the following improvements have been introduced in the Data Insights Generator component:

- Lite report generation: A new option to generate lighter versions of the reports have been introduced. In order to improve the response time of those components using the Data Insights Generator component, a faster way to generate reports (leaving out the processing that might require a higher time/computational cost, such as correlations) has been deployed.
- CSV file separator identification has been improved.

[5] https://doc.piveau.io/metrics/

2.7.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data Provider	Get insight from my dataset	Get an insight information from a dataset	Alpha	Insight data generated with the initial set of insights	Done	PISTIS.OUS.03
UC_02	Data Provider	Get an improved set of insights from my dataset	Get an extended insight information from a dataset	Beta	Insight report generated with an extended set of insights	Done	PISTIS.OUS.03
UC_03	Data Provider	Get the final set of insights from my dataset	Get the final set of insights from a dataset	v1.0	Insight report generated with the final set of insights	Upcoming	PISTIS.OUS.03

2.7.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	The component has to extract/generate some information describing the data (metadata) by analysing the data.	UC_01, UC_02, UC_03, UC_04	

2.7.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Compatibility	NFR1	The component has to be able to analyse data in CSV format (Pandas DataFrame compliant).
Security	NFR2	PISTIS ensures that only authorised user can get the insights.

2.7.6 Component Architecture

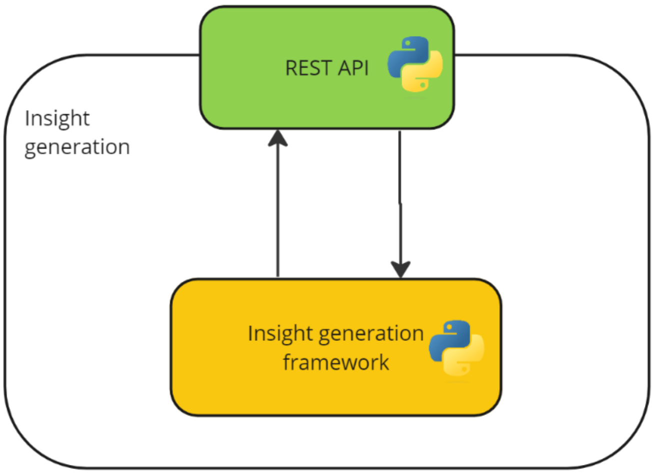


Figure 28: Component's Internal Architecture

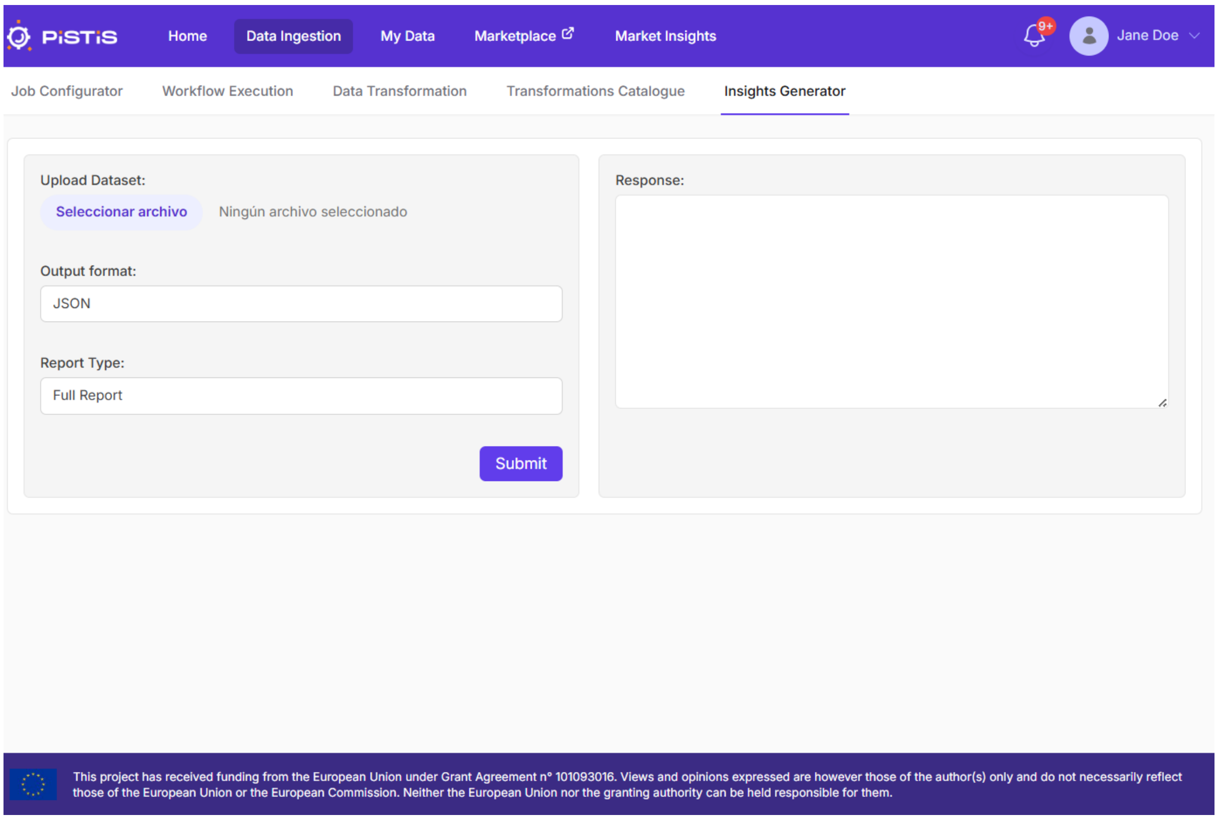
The component consists of a tool that when getting an input dataset via API, returns a JSON with some insights of that input dataset. This API module has been built using Flask, while the logic behind the insight generation itself relies on the ydata profiling library.

2.7.7 Technology Background

The insight generation component relies on python libraries for data validation (by means of a widely used python library for data manipulation such as pandas) and for the insight generation report creation.

2.7.8 Graphical User Interface

As we can see in the following screenshot, the current version of the Data Insights Generator provides the mechanisms to choose the output format of the report (json or html) and the type (full version, slower to generate or lite and faster version) of the report to generate.



Upload Dataset: Ningún archivo seleccionado

Output format: JSON

Report Type: Full Report

Response:

This project has received funding from the European Union under Grant Agreement n° 101093016. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Figure 29: Data Insights Generator Component UI in beta version

2.7.9 Developer Documentation

The Data Insights generator component documentation is available at the following URL: https://docs.pistis-market.eu/developers/insights-generator/insights-generation

2.7.10 Source Code

The source code developed for the Insight Generator component is currently stored under an internal repository in ATOS premises until the license to be applied to it is decided.

The component’s UI (frontend) is part of the platform-frontend repository and its code is maintained in GitHub under the PISTIS project and can be accessed via the respective link: https://github.com/PISTIS-Platform/platform-frontend

3 Data & Metadata Storage Bundle

The Data & Metadata Storage bundle is delivering a catalogue for the data that are made available by each organisation in their own PISTIS Data Factory environment. Moreover, it also concerns those made available as "published" datasets over the whole ecosystem, alongside with the appropriate data storage facilities to hold the data.

This bundle consists of the following components:

- Data Catalogues
- Factory Data Storage

These are presented in the following sub-sections.

3.1 Data Catalogues

3.1.1 Component Description

Under "data catalogues" we refer to the components that offer catalogue features for the data in PISTIS. They constitute the essential components to manage the offerings of data assets and are the following:

- **Factory Data Catalogue**
- **PISTIS Data Catalogue**

The **Factory Data Catalogue** operates at the data provider's site, serving as the gateway to an organisation's data assets. It facilitates the management and availability of both metadata and data, with each organisation responsible for maintaining its own catalogue and incorporating their own (meta)data. In contrast, the **PISTIS Data Catalogue** is a centralized marketplace service within the PISTIS Cloud Platform and aggregates the metadata from every Factory Data Catalogue. As a marketplace, it enables users to explore diverse data assets shared by trusted data providers.

The Factory Data Catalogues and the PISTIS Data Catalogue are interconnected through the Asset Description Bundler, which publishes only the relevant metadata needed for acquisition on the PISTIS Cloud Platform. This mechanism ensures that while the metadata is accessible, the actual data remains with the data providers until a transaction is executed.

3.1.2 Main Improvements in Beta Version

The latest improvements in this beta release are:

The unified GUI header has been implemented, which enhances user experience by providing seamless navigation across PISTIS components. This improvement reduces visual disruptions and gives a more cohesive feel throughout the platform.
The look and feel have been improved for better navigation and usability, and distributions now can be downloaded.

Data providers can customize their data model according to their needs by using Shapes Constraint Language (SHACL).

3.1.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data Provider	Provide metadata for my data	I can increase my data quality	Alpha	CRUD operations can be done to the selected metadata	Done (delivered in alpha)	PISTIS.OUS.01
UC_02	Data Consumer	Find the data I need by searching in the metadata using keywords and filters	I can reach my goal	Alpha	Search function returns the most relevant datasets	Done (delivered in alpha)	PISTIS.OUS.09
UC_03	Data Consumer	See what is available	I can brainstorm new ideas or make a reliable prototype	Alpha	All datasets are listed	Done (delivered in alpha)	PISTIS.OUS.09
UC_04	Data Provider	Configure custom data schema for my data	I can customise my data schema according to my requirement	Beta	If needed, having a customised data schema is possible	Done	PISTIS.OUS.01
UC_05	Data Provider	See what other Data Providers offer	I know the competition/ market, if I want to make money by selling data	v1.00	Data from all providers are listed in the PISTIS Data Catalogue	Upcoming	PISTIS.OUS.09
UC_06	Data Consumer	Get/buy the data I need	I can start working on my goal	v1.00	The required information to get/buy a data is presented	Upcoming	PISTIS.OUS.10

3.1.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	Create, read, update, delete (meta)data.	US_01, US_04, US_06	
FR_02	Search metadata and filter the result.	US_02, US_03, US_05	

3.1.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Functional Suitability	NFR1	The Data Catalogue complies with all specified functional requirements.
Performance efficiency	NFR2	The Data Catalogue is capable in handling a high volume of metadata CRUD operations while maintaining its stability and performance.
Compatibility	NFR3	The Data Catalogue features a REST API service, which has become the standard for software services integration. This means it is compatible with all other components capable of sending REST API requests and processing the received responses.
Usability	NFR4	Accompanied by detailed API documentation, it allows users to quickly understand all available endpoints. When a request fails, an error message is automatically generated to help the users to solve the issues.
Reliability	NFR5	The Data Catalogue consistently gives responses that accurately correspond to the given requests.
Security	NFR6	An authentication procedure can be configured in the Data Catalogue to enable the catalogue owner to grant access only to authorized users.
Portability	NFR7	The Data Catalogue supports containerisation. When needed, it can be deployed using container technologies like Docker and orchestrated with systems like Kubernetes.

3.1.6 Component Architecture

Both the Factory and PISTIS Data Catalogues are based on Piveau, an open-source, Java-based data management solution. The data management solution is represented by piveau-hub in Figure 30.

The Data Catalogue (piveau-hub) primarily consist of two main services: the repository service, which manages RDF metadata in accordance with the DCAT-AP standards, and the search service, which allows users to efficiently find the metadata they need. When a metadata is added, it is initially processed by the repository service and stored in a Triplestore database, Virtuoso. After that, the repository service converts the metadata into JSON format and transfers it to the search service, enabling it to be indexed, stored, and managed within Elasticsearch.

The repository and search services each have a dedicated API that can be utilized by any PISTIS component capable of sending REST API requests and processing the responses. Consequently, these APIs are essential for facilitating integration with other PISTIS components. Additionally, a frontend is provided to facilitate user interaction with the Data Catalogues.

The Factory and PISTIS Data Catalogue are kept in sync through the Asset Description Bundler component, which operates independently from the core Data Catalogue component and is not part of the Data Catalogue. Another component that interacts with the Factory Data Catalogue is the Organisational Metadata and Data Provider and Consumer component. Meanwhile, the PISTIS Data Catalogue interacts with PISTIS IAM, Data Model Repository, and

PISTIS Cloud Platform Metadata Consumer components (for example, Smart Contract Execution Engine).

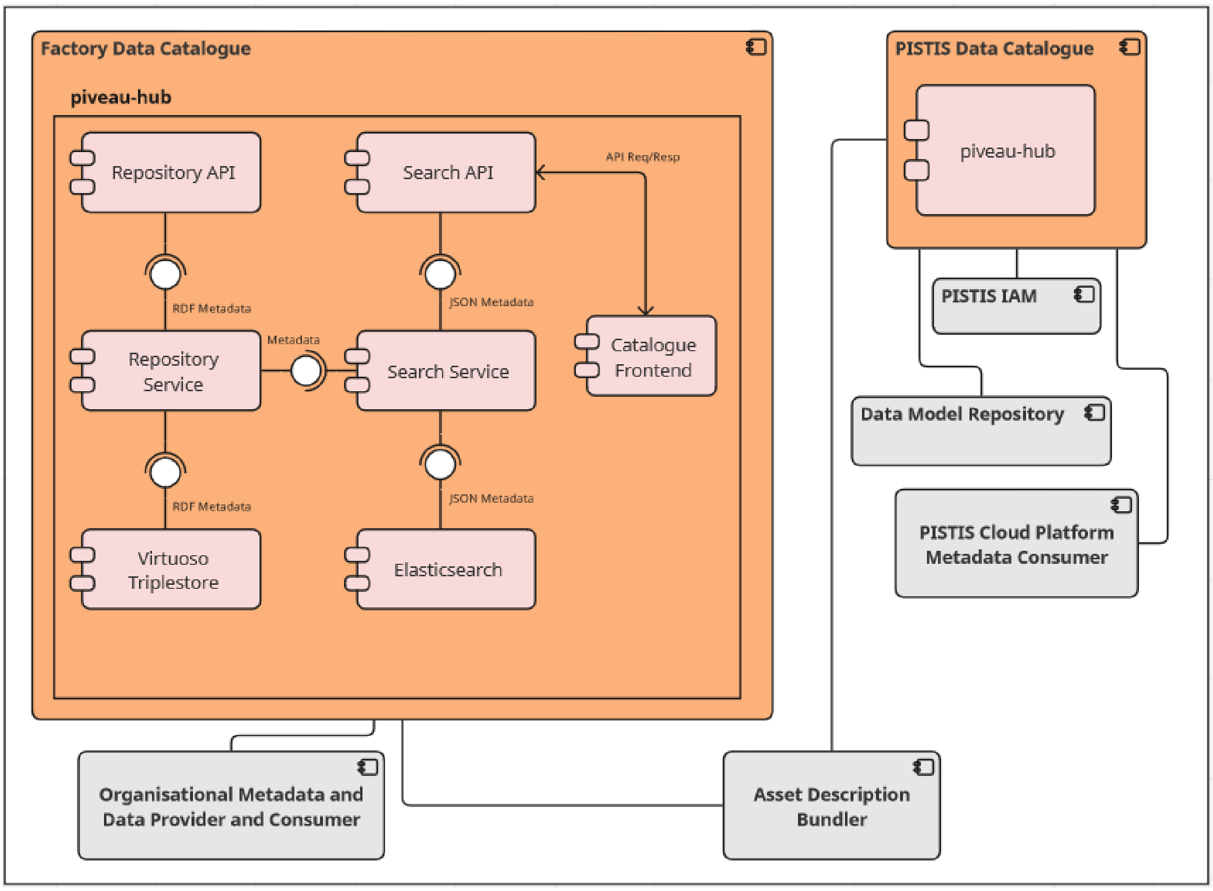


Figure 30: Data Catalogue's Internal Architecture

3.1.7 Technology Background

The Data Catalogue will be based on the scalable, Open Source and Java-based metadata management solution piveau[6]. Piveau is a catalogue solution, designed around Semantic Web technologies and applies a Triplestore as its primary database to leverage the full potential of Linked Data. That allows to store metadata, data and data models in native RDF (Resource Description Format) without any restrictions. Especially, the integration of external existing data via the principles of Linked Data is covered by the solution. It closes the gap between formal Linked Data metadata specifications and their actual application in production. The base data model is DCAT, but it can be extended to support any possible data model via providing suitable Shapes Constraint Language (SHACL) files. Piveau is designed to harmonize metadata from various sources into a single harmonized knowledge graph by applying a common URI schema to all incoming data.

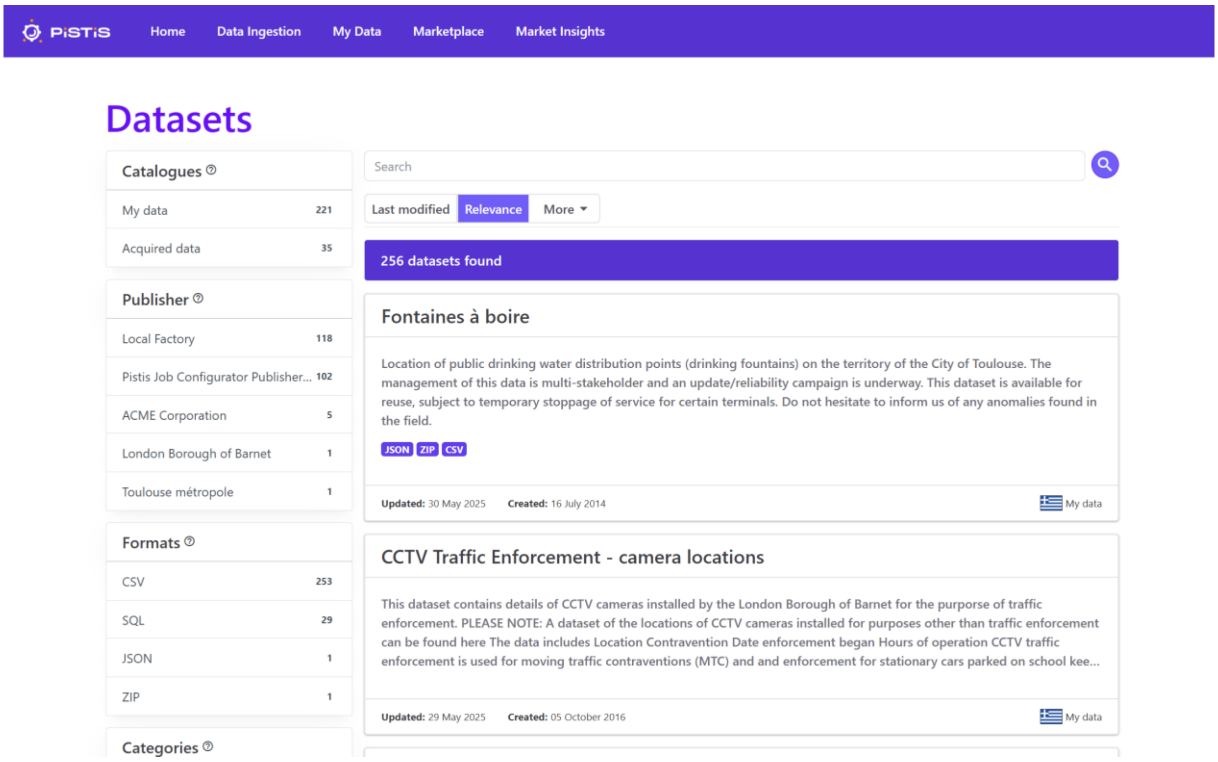
[6] https://doc.piveau.io

To help users quickly find the datasets they need, Piveau integrates a high-performance search service based on Elasticsearch. Furthermore, Piveau comes with a user-friendly interface, developed with JavaScript and Vue.js, that simplifies metadata browsing and discovery.

Piveau is already prepared to be integrated with Keycloak for access control and can be deployed out-of-the-box on cloud infrastructures, like Kubernetes.

3.1.8 GRAPHICAL USER INTERFACE

Figure 31 displays a Data Catalogue listing all datasets, with filtering options on the left-hand side to help users in narrowing the list. Users can filter the dataset list or search result, if needed, based on specific criteria or metrics such as data publisher, data format, relevant keywords or metadata quality.



Datasets

Catalogues	
My data	221
Acquired data	35

Publisher	
Local Factory	118
Pistis Job Configurator Publisher...	102
ACME Corporation	5
London Borough of Barnet	1
Toulouse métropole	1

Formats	
CSV	253
SQL	29
JSON	1
ZIP	1

Categories

256 datasets found

Fontaines à boire

Location of public drinking water distribution points (drinking fountains) on the territory of the City of Toulouse. The management of this data is multi-stakeholder and an update/reliability campaign is underway. This dataset is available for reuse, subject to temporary stoppage of service for certain terminals. Do not hesitate to inform us of any anomalies found in the field.

JSON ZIP CSV

Updated: 30 May 2025 **Created:** 16 July 2014 My data

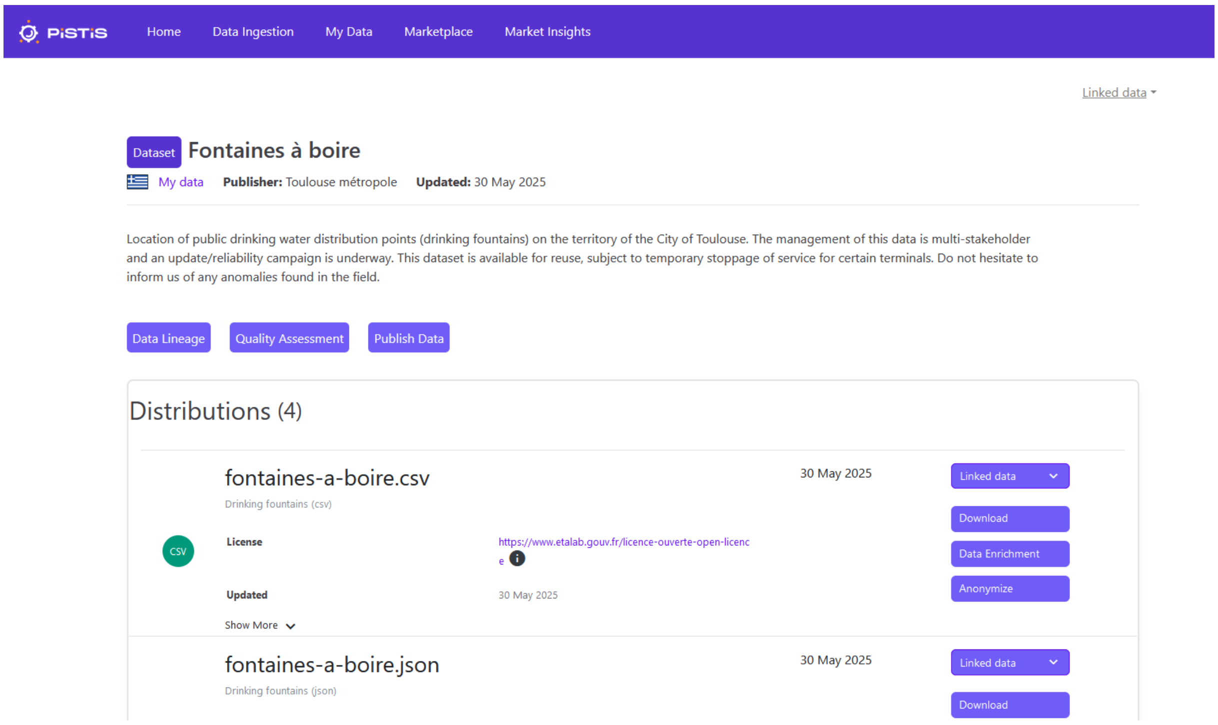
CCTV Traffic Enforcement - camera locations

This dataset contains details of CCTV cameras installed by the London Borough of Barnet for the purporse of traffic enforcement. PLEASE NOTE: A dataset of the locations of CCTV cameras installed for purposes other than traffic enforcement can be found here The data includes Location Contravention Date enforcement began Hours of operation CCTV traffic enforcement is used for moving traffic contraventions (MTC) and and enforcement for stationary cars parked on school kee...

Updated: 29 May 2025 **Created:** 05 October 2016 My data

Figure 31: A list of datasets with filters on the left-side

Figure 32 shows the dataset detail page in a Factory Catalogue, where data asset owners can track dataset lineage and assess its quality. This page is also the entry point to publish a data asset to the PISTIS Data Catalogue. For each distribution, data asset owners can enrich and anonymize the data as needed.



Fontaines à boire

My data **Publisher:** Toulouse métropole **Updated:** 30 May 2025

Location of public drinking water distribution points (drinking fountains) on the territory of the City of Toulouse. The management of this data is multi-stakeholder and an update/reliability campaign is underway. This dataset is available for reuse, subject to temporary stoppage of service for certain terminals. Do not hesitate to inform us of any anomalies found in the field.

Distributions (4)

fontaines-a-boire.csv — 30 May 2025

Drinking fountains (csv)

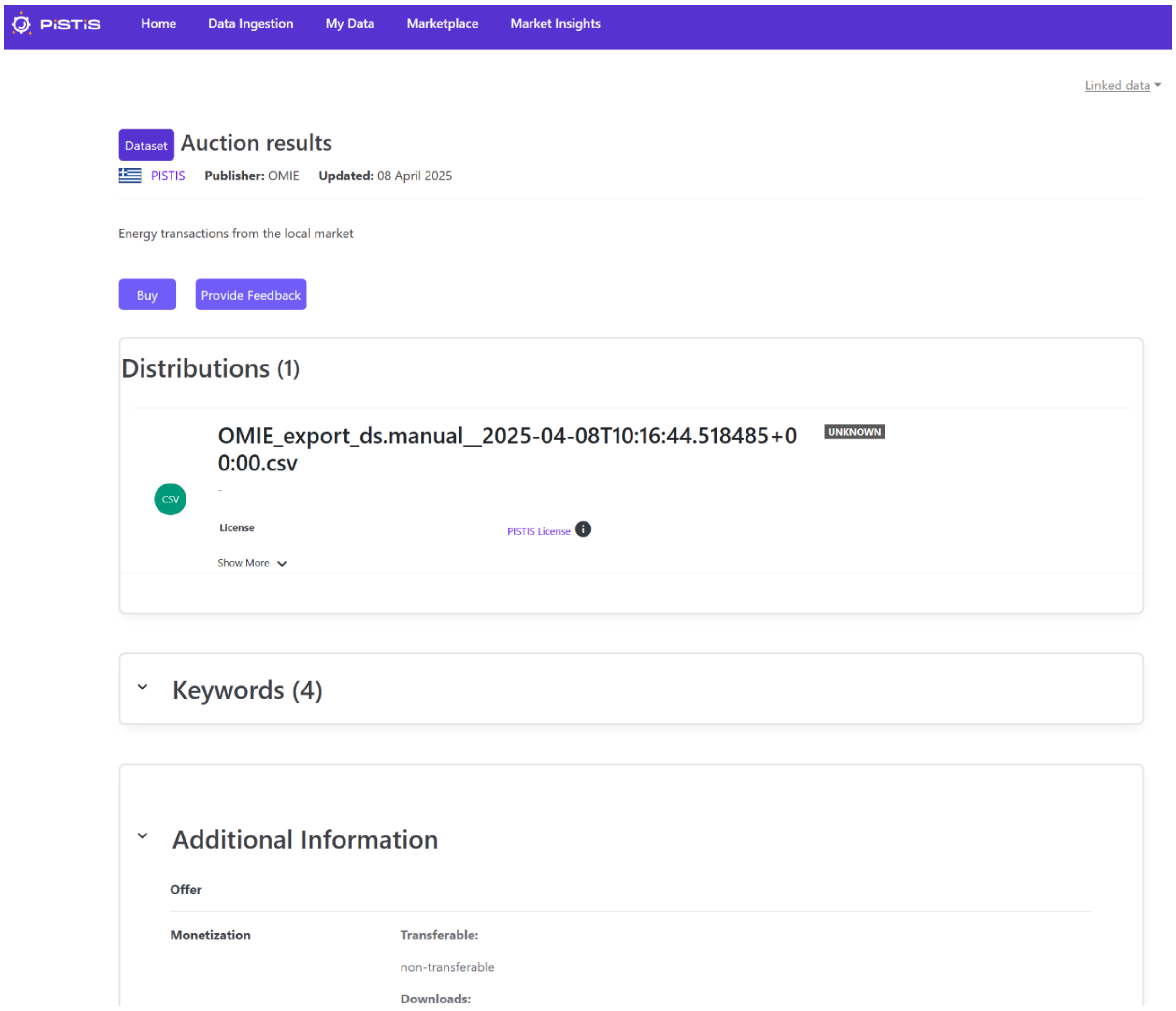
License	https://www.etalab.gouv.fr/licence-ouverte-open-licence
Updated	30 May 2025

fontaines-a-boire.json — 30 May 2025

Drinking fountains (json)

Figure 32: Presentation of a dataset and its associated metadata

Figure 33 displays the dataset detail page in the PISTIS Dataset Catalogue, serving as the entry point for data consumers to purchase a data asset or provide feedback.



Auction results

PISTIS **Publisher:** OMIE **Updated:** 08 April 2025

Energy transactions from the local market

Distributions (1)

OMIE_export_ds.manual__2025-04-08T10:16:44.518485+00:00.csv

License PISTIS License

Keywords (4)

Additional Information

Offer

Monetization Transferable: non-transferable

Downloads:

Figure 33: Dataset detail page in the PISTIS Dataset Catalogue

3.1.9 Developer Documentation

Documentation and guides are available at https://docs.pistis-market.eu/developers/factory-catalogue/introduction. They provide support for implementation and integration with other components. Key topics covered include entity creation and distribution management.

3.1.10 Source Code

The data catalogues (Factory and Cloud) are built using the open source solution piveau-hub[7]. A list of used services can be found in the corresponding GitHub repository[8]. Hub-repo acts as a middleware and abstraction layer to interact with the Triplestore. It offers a RESTful interface, supporting the major RDF serializations (Turtle, JSON-LD, N-Triples, RDF/XML,

[7] https://gitlab.com/piveau/hub

[8] https://github.com/PISTIS-Platform/pistis-catalog-backend

Notation3). Hub-search is responsible for encapsulating the communication between hub-repo and search engine (Elasticsearch) for enabling full-text search. The frontend is developed with Vue.js and the codes can also be found on PISTIS GitHub repository[9].

3.2 Factory Data Storage

3.2.1 Component Description

The Factory Data Storage is the core storage component for PISTIS factories, responsible for storing both raw and processed datasets. It operates locally within the Data Factory environment, which resides on-premises at organizations participating in the PISTIS ecosystem. Data from each organization is ingested into this storage system via the Data Check-In module. The storage system includes two main databases:

- A primary database for relational datasets stored as SQL tables.
- A secondary database for storing datasets as files.

Access to both databases is provided through a RESTful API. In the alpha release, PostgreSQL was used to store both SQL tables and files. However, in the beta version, MongoDB replaces PostgreSQL for file storage due to its superior performance with large files. The choice of database depends on the dataset format. Relational datasets with a defined schema are stored as SQL tables in the primary database. Each table is assigned a unique UUID, which can be used to query its rows and columns. File-based datasets (e.g., .csv, .txt, .xml) are stored in the MongoDB database, also identified by a UUID. These files can later be transformed into SQL tables and migrated to the primary database.

The main functionalities of the Factory Data Storage are:

- Storage for tables in a relational database and files in MongoDB
- Access to the database using a REST API
- CRUD operations on tables
- CRUD operations on files

3.2.2 Main Improvements in Beta Version

In the beta release of this component, the primary focus was on enhancing performance and extending support for a broader range of file formats, making the service more scalable and performant. Significant improvements were made to ensure the system can efficiently handle larger datasets and support seamless integration with other components of the PISTIS

[9] https://github.com/PISTIS-Platform/pistis-catalog-ui

ecosystem. One of the key changes was the transition from PostgreSQL to MongoDB for file storage. MongoDB was selected for its superior performance in handling large, unstructured files and its native support for flexible data formats, which aligns well with PISTIS's evolving data ingestion needs. In addition, robust error handling mechanisms were introduced. These enhancements improve system stability, offer clearer feedback in case of failures, and facilitate smoother interoperability with other PISTIS modules. This not only helps developers diagnose issues more effectively but also enhances the overall user experience by reducing unexpected disruptions.

Key improvements in the beta release are:

- Migration to MongoDB for storing files, replacing PostgreSQL
- Implementation of comprehensive error handling to support better integration and user interaction.

3.2.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data provider	Store a dataset in the form of a file	It is made available in the PISTIS platform	Alpha	The format and size of the file is supported by the data storage	Done	PISTIS.OUS.01, PISTIS.OUS.06
UC_02	Data provider	Store a dataset in the form of a file or a table	It is made available in the PISTIS platform	Alpha	The data schema is correctly defined	Done	PISTIS.OUS.01
UC_03	Data provider	Fetch a dataset in the form of a file	It can be transformed into a tabular dataset with a domain specific data model	Alpha	Has access rights to the file	Done	PISTIS.OUS.03
UC_04	Data provider	Fetch a dataset in the form of a table	It can be used for any of the other factory components for quality check or data transformation	Alpha	Has access rights to the table	Done	PISTIS.OUS.04
UC_05	Data provider	Update a dataset in the form of a table or a file	It can be made available for further processing	Alpha	Has access rights to the table	Done	PISTIS.OUS.04

			and quality analysis				

3.2.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	The Factory Data Storage stores datasets in the form of SQL tables and files.	US_01, US_02	
FR_02	Datasets in the form of files and tables can be read from the Factory Data Storage.	US_03, US_04	
FR_03	Datasets in the form of files and tables can be updated in the Factory Data Storage.	US_05	

3.2.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Functional Suitability	NFR1	The Factory Data Storage is built respecting all functional requirements.
Performance efficiency	NFR2	The Factory Data Storage API stores files and tables without causing delay.
Compatibility	NFR3	The Factory Data Storage API can be integrated with other components of the PISTIS Factory Architecture.
Usability	NFR4	All endpoints of the Factory Data Storage adhere to the standards of OpenAPI and are functional with proper configuration.
Reliability	NFR5	All endpoints of the Factory Data Storage are always functional and proper error messages are provided when a request fails.
Security	NFR6	The Factory Data Storage will be protected with the Identity and Authorization Manager and access policies.
Portability	NFR7	The Factory Data Storage is containerized and can run on any hardware that supports docker.

3.2.6 Component Architecture

The Factory Data Storage is built using a Python Flask app and two databases, PostgreSQL and MongoDB to meet PISTIS's factory storage requirements. The Flask framework provides the infrastructure for handling HTTP requests and defining endpoints, facilitating communication between the clients, backend and the database. It builds endpoints to create, read, update, and delete data with ease. In PISTIS, two types of data are being handled using the storage, data in the form of structured tables with a specific data schema and data in the form of files. The API of the Factory Data Storage provides separate POST endpoints to upload files and to create tables. Along with these POST endpoints, GET, PUT and DELETE endpoints are also built to retrieve, update and delete data in the database. For data stored as tables, the update operation will add rows to a table and for data stored as files, the update operation can update the file or rename a file. The input to these endpoints will vary depending on the type of the request and the type of the data being handled by the request. For example, files are uploaded using a formdata parameter “File” and tables are created using a JSON body with the data model, data and any metadata such as the name of the table.

Inputs provided to every tables endpoint is transferred into an SQL query or a python object and is used to interact with the PostgreSQL database using SQLAlchemy. SQLAlchemy's Object-Relational Mapping (ORM) capabilities allow interaction with the Postgres relational database using Python objects along with raw SQL queries. Relational tables with fixed data schema can be created by defining them as a Python object, and if the schema is not known, then native SQL queries are created and ran on the database using the SQLALchemy engine.

PostgreSQL is the chosen relational database to handle SQL tables, which is a reliable and scalable database management system that stores and manages data with high efficiency. Data inserted into the PISTIS factory through the Data Check-in process are separated based on their format and is stored in either of the two databases (Postgres or Mongo) for tables or files. Every dataset stored in the database has a UUID assigned to it, this UUID is the unique identifier of a dataset and is used to perform GET, UPDATE and DELETE operations on this dataset. In MongoDB, files up to 16MB can be stored as documents and anything larger is stored using GridFS. GridFS splits a file into smaller chunks and stores them as a metadata collection and binary data collection. The *gridfs* module in PyMongo is used for interaction of the Factory Data Storage backend with MongoDB.

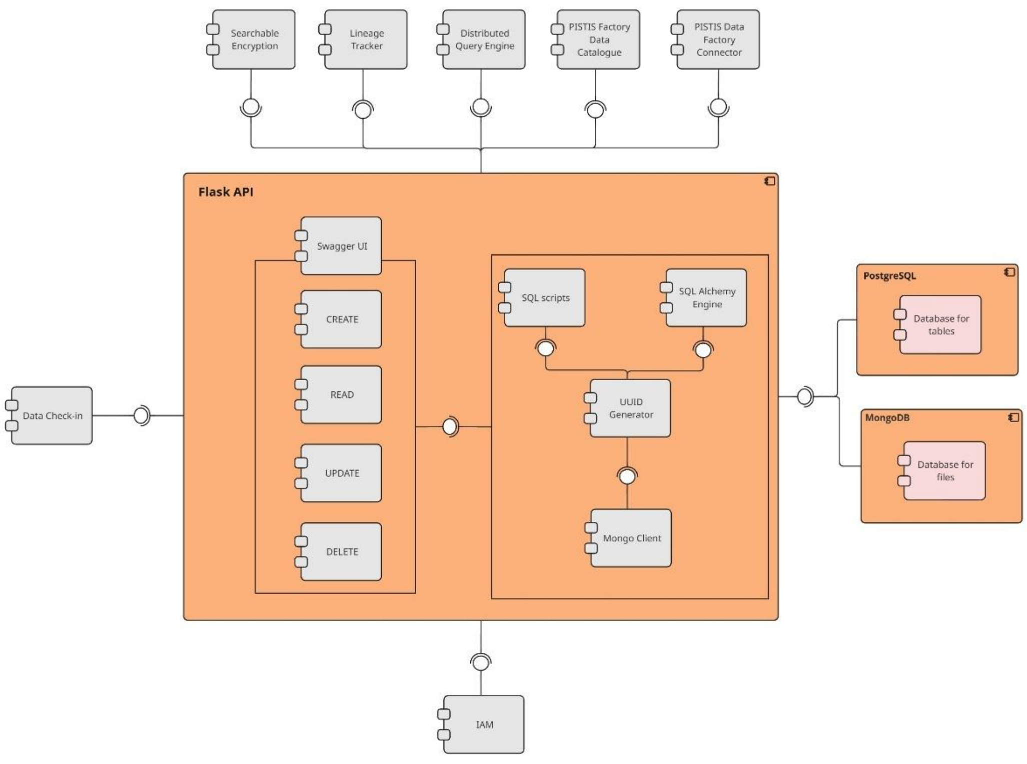


Figure 34: Factory Data Storage Internal Architecture

3.2.7 Technology Background

Factory Data Storage consists of two core modules: the databases for storing files and the RESTful API for accessing them. It uses two types of databases—PostgreSQL, an open-source object-relational database management system, and MongoDB, a scalable, document-oriented NoSQL database suited for handling both structured and unstructured data. The REST API is built using Python Flask, and extended with:

- SQLAlchemy, a powerful Python SQL toolkit and Object Relational Mapper (ORM) for interacting with PostgreSQL.
- PyMongo, the official MongoDB driver for Python, which facilitates communication with the MongoDB instance.

The PostgreSQL and MongoDB databases run as separate instances, but are seamlessly integrated and accessed through the unified RESTful API.

3.2.8 Graphical User Interface

This is a backend service, and has no UI.

3.2.9 Developer Documentation

The Factory Data Storage component of the PISTIS platform manages and provides access to both structured and unstructured datasets via a RESTful API connected to two databases. The developer documentation of this service is available at: https://docs.pistis-market.eu/developers/factory-data-storage/intro.

3.2.10 Source Code

The source code of Factory Data Storage is available at https://github.com/PISTIS-Platform/factory-data-storage and the Swagger UI is accessible through this URL: https://develop.pistis-market.eu/srv/factory-data-storage/.

4 DATA DISCOVERY BUNDLE

The Data Discovery bundle provides services for searching and discovering the available data assets that might be of interest for a Data Consumer.

This bundle consists of the following components:

- Distributed Query Engine
- Matchmaking Services

Distributed Query Engine is presented in the following sub-sections. Matchmaking Services are presented in D3.2.

4.1 DISTRIBUTED QUERY ENGINE

4.1.1 COMPONENT DESCRIPTION

The main purpose of this component is to query directly the unstructured or semi-structured data to discover datasets that cannot be retrieved by querying their metadata on the Distributed Data Catalogue.

However, the volume of the data stored in the Data Factories does not allow extensive search approaches to be used. Therefore, Locality Sensitive Hashing techniques are employed to quickly obtain a list of matches. Subsequently, the lists of potential matches yielded by the LSH modules that reside in every Data Factory are further evaluated and enriched with metadata retrieved by the Distributed Data Catalogue.

Then, the merged list is cross-checked with Keycloak to decide if the users that performed the search have access rights to the results. The matched datasets for which the access is forbidden are filtered out. Finally, this list is fed to a collection of pretrained ML-models that re-rank it based on the datasets’ metadata in order to give prominence to the most relevant matches.

4.1.2 MAIN IMPROVEMENTS IN BETA VERSION

The beta release of the information retrieval system introduces two key architectural upgrades over the alpha version. The core enhancement is the deployment of a new ML-based re-ranker module, which processes the candidate document set from the initial first-pass retrieval performed on the Data Factories. This component uses a fitted model to re-score the merged list of results, significantly boosting top-k precision by better evaluating relevance based on additional information fetched from the Distributed Data Catalogue. On the front end, the Distributed Query endpoints have been fully integrated into PISTIS main GUI.

4.1.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	User	To search for dataset	I can discover datasets based on stored data	Alpha	The user can perform queries using an API	Done	PISTIS. OUS.9
UC_02	Distributed Query backend	Create forwarding service	To forward the metadata queries to the Data Catalogue	Alpha	Metadata queries are automatically sent to the Data Catalogue	Done	PISTIS. OUS.9
UC_03	LSH	Create an LSH service	Datasets are indexed for enabling quick NN queries	Alpha	Datasets can be indexed and retrieved by making API calls	Done	PISTIS. OUS.9
UC_04	ReRanker	Create Merger/ReRanker	The results returned by the Data Catalogue and the LSH service are unified	Alpha	A unified list of results that match both types of searching is returned	Done	PISTIS. OUS.9
UC_05	ReRanker	Create ReRanking training service	To fit a model that will help predict a more accurate ranking of the results	Beta	The list of US_4 is sorted based on relevance	Done	PISTIS. OUS.9
UC_06	User	To search for dataset using GUI	I can discover datasets based on stored data in a user-friendly manner	Beta	The user can perform queries and browse results using a GUI	Done	PISTIS. OUS.9

4.1.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	The users of the PISTIS platform must be able to search for datasets based on their contents.	US_01, US_03, US_06	
FR_02	The component should support queries with text or binary data.	US_03	
FR_03	The component should support queries over streaming data.	US_03	
FR_04	The PISTIS platform will have a unified UI for searching datasets based both on data and metadata (see PISTIS Data Catalogue).	US_01, US_02, US_03, US_04,	

		US_05, US_06	
FR_05	Filter out those results that the user does not have read access for.	US_01	

4.1.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Performance efficiency	NFR1	The overall process shall be performed without delays and should not consume unnecessary system resources.
Reliability	NFR2	The Distributed Query Engine shall operate in a reliable manner, checking efficiently the connection status of the PISTIS Data Factories in the network.
Security	NFR3	The overall process shall be made through secure communication channels.
Usability	NFR4	The GUI for searching datasets shall be intuitive and user-friendly.

4.1.6 Component Architecture

The Distributed Query Engine is divided into three major subcomponents: the Distributed Query Service, the Locality Sensitive Hashing component and the ReRanker. At its heart lies the Distributed Query Service which orchestrates the various tasks that need to be executed. First, it receives the search requests performed by the end users and forwards them to the LSH components that run on every Data Factory. When the subsequent searches are finished, it gathers all the results in the form of lists of datasets' UUIDs and communicates with the IAM component to check whether the users have the appropriate access rights over them. Then it retrieves the datasets' details and metadata from the Data Catalogue. Finally, before these results are returned to the end users, they are sorted by the ReRanker component. As it has already been mentioned, the LSH component is present on every Data Factory and is charged with indexing the datasets that are stored in the Factory's Data Storage. It comprises of four modules: the Index Creator that generates the hashes that describe a dataset and is triggered whenever a dataset is inserted/updated, the Hashes Storage where the aforementioned hashes are persisted, the Query Executor that is charged with retrieving the most relevant datasets given some query data and a Controller that manages all the above and exposes an API to the rest of the PISTIS components. The last subcomponent of the Distributed Query Engine is the ReRanker which performs the task of merging and re-arranging the various lists of results. To achieve this, it has two modules that employ ML techniques: the first one is for training a model that will perform the sorting and the second one uses the fitted model to predict a more accurate ranking of the unified list of results.

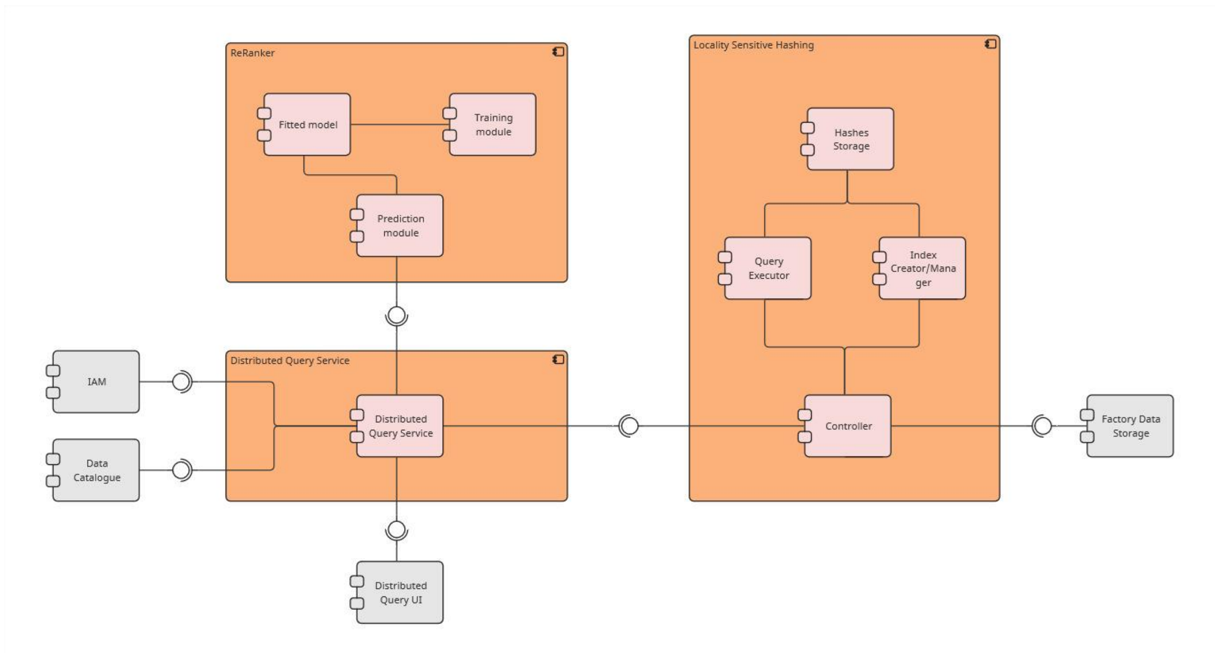


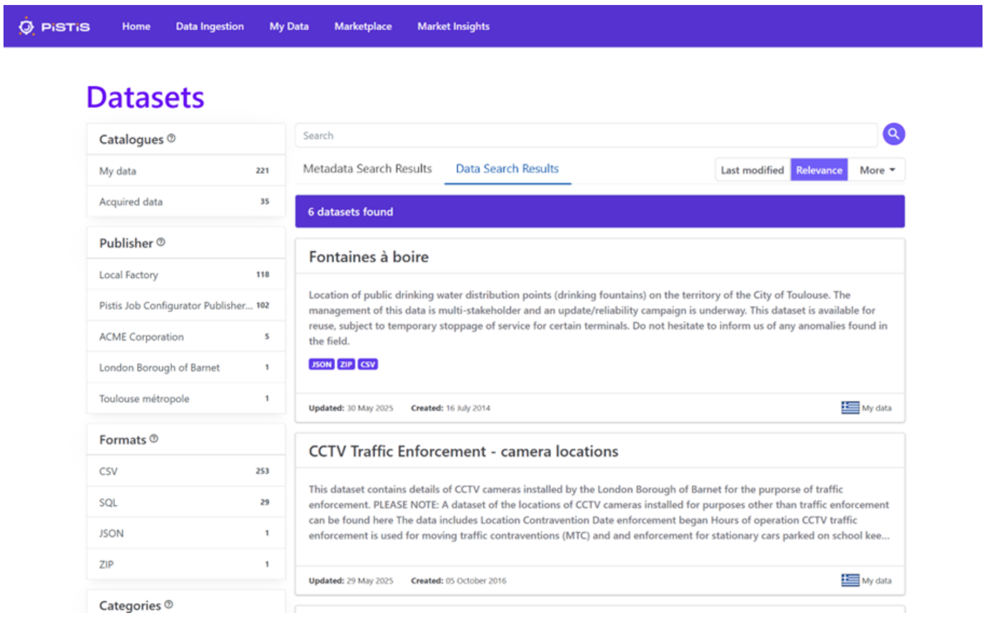
Figure 35: Distributed Query Engine Internal Architecture

4.1.7 Technology Background

The following main technologies are employed for developing the Distributed Query Engine:

- The Redis NoSQL database is used for storing and retrieving the hashed generated by the LSH component of the Distributed Query Engine.
- Python's Flask microframework has been employed for creating all the web APIs exposed by Distributed Query Engine's services to the rest of the PISTIS components.
- For developing the ML part of the ReRanker various Python libraries have been used. Some of the most notable among them are: numpy for handling numerical data, scikit-learn for data preprocessing and classification using classic algorithms and pytorch for building and training Neural Networks etc.

4.1.8 Graphical User Interface



Datasets

Catalogues	
My data	221
Acquired data	35

Publisher	
Local Factory	118
Pistis Job Configurator Publisher...	102
ACME Corporation	5
London Borough of Barnet	1
Toulouse métropole	1

Formats	
CSV	253
SQL	29
JSON	1
ZIP	1

Categories

6 datasets found

Fontaines à boire

Location of public drinking water distribution points (drinking fountains) on the territory of the City of Toulouse. The management of this data is multi-stakeholder and an update/reliability campaign is underway. This dataset is available for reuse, subject to temporary stoppage of service for certain terminals. Do not hesitate to inform us of any anomalies found in the field.

JSON ZIP CSV

Updated: 30 May 2025 **Created:** 16 July 2014 My data

CCTV Traffic Enforcement - camera locations

This dataset contains details of CCTV cameras installed by the London Borough of Barnet for the purpose of traffic enforcement. PLEASE NOTE: A dataset of the locations of CCTV cameras installed for purposes other than traffic enforcement can be found here The data includes Location Contravention Date enforcement began Hours of operation CCTV traffic enforcement is used for moving traffic contraventions (MTC) and and enforcement for stationary cars parked on school kee...

Updated: 29 May 2025 **Created:** 05 October 2016 My data

Figure 36: Searching for datasets using the GUI

The "Data Search" tab is used to perform queries directly against the data content, rather than its metadata. Users may enter their queries into the search bar at the top of the page.

4.1.9 Developer Documentation

The developer documentation of this service is available at:

https://docs.pistis-market.eu/developers/distributed-query/distributed-query

4.1.10 Source Code

The source code of the component is maintained in GitHub as a git repository under the PISTIS project and can be accessed via the respective link: https://github.com/PISTIS-Platform/distributed-query.

5 Data Exchange Bundle

The Data Exchange bundle facilitates the peer-to-peer exchange of the data assets between a Data Provider and a Data Consumer, adhering to the terms of the contract that has been signed to govern the overall transaction.

This bundle consists of the following components:

- PISTIS Data Factory Connector
- Smart Contract Checker

These are presented in the following sub-sections.

5.1 PISTIS Data Factory Connector

5.1.1 Component Description

The transfer of data between different PISTIS actors that belong to different organisations (e.g. Data Providers and Data Consumers) is the logical termination point of a monetary or otherwise exchange agreement flow, where following the establishment of an electronic contract, the dataset that is part of the agreement must reach the Data Consumer.

The overall transfer in PISTIS is facilitated by the PISTIS Data Factory Connector (or else, the PISTIS Connector), which is an infrastructure that is tasked, once a data transfer contract needs to be executed, to fetch the data stored in the PISTIS Data Factory of the Data Provider and pass it to the PISTIS Data Factory of the Data Consumer.

This transfer is to be performed following the appropriate checks at smart contract level that will govern such exchanges (based on licence, usage and permission attributes stored in the ledger), and the result will be the deposition of the data asset purchased by the Data Consumer in his own, local data storage.

5.1.2 Main Improvements in Beta Version

In this version, the Data Factory Connector, apart from the one-off sale of a dataset (alpha version), supports also:

- the transfer of a dataset from the Data Factory of the Provider to the Data Factory of the Consumer that has been acquired with a paid subscription (static dataset or a dataset updated based on a schedule, and
- the subscription of the data owner to a Kafka stream that is sold by a data owner.

5.1.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component. UC_09 has been added to that list, as a new feature.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data Consumer	Have a data asset I've already bought, automatically transferred to my Data Factory	I can use the dataset on my side	Alpha	The data asset bought is in the Data Consumer Storage	Done	PISTIS. OUS.1 2
UC_02	Data Consumer	Have a dataset to which I paid a subscription to automatically be updated in my Data Factory	I can use the dataset on my side	Beta	The data asset bought is in the Data Consumer Storage	Done	PISTIS. OUS.1 2
UC_03	Data Consumer	Have a slice of the dataset I bought automatically transferred to my Data Factory	I can use the dataset on my side	V1.00	The data asset bought is in the Data Consumer Storage	Upcom ing	PISTIS. OUS.1 2
UC_04	Data Provider	Log all data transfers I made to Buyers in the blockchain	There is evidence and information about those	Alpha	The Ledger contains data transfer logs	Done	PISTIS. OUS.1 2
UC_05	Data Consumer	Get notified once a transfer has finished	I can use the dataset on my side	V1.00	Notifications of executed transfers are shown to Data Consumer	Upcom ing	PISTIS. OUS.1 2
UC_06	Data Consumer	Get notified once a transfer has failed and be provided with an error code	I can contact PISTIS to get support	V1.00	Notifications of failures are shown to Data Consumer	Upcom ing	PISTIS. OUS.1 2
UC_07	Data Provider	Get notified once a transfer has finished	I know the Data Consumer got his purchase	V1.00	Notifications of executed transfers are shown to Data Provider	Upcom ing	PISTIS. OUS.1 2
UC_08	Data Provider	Get notified once a transfer has failed and be provided with an error code	I can contact PISTIS to get support	V1.00	Notifications of failures are shown to Data Provider	Upcom ing	PISTIS. OUS.1 2
UC_09	Data Consumer	Be able to connect and ingest data from a stream to which I bought access	These data can become instantly available at my side	Beta	Streaming data available in the Data Factory of the Consumer	Done	PISTIS. OUS.1 2

5.1.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	The PISTIS Data Factory Connector shall transfer the acquired data asset (either static or streaming data) from the PISTIS Data Factory of the Data Provider to that of the Data Consumer.	UC_01, UC_02, UC_03, UC_04	N/A
FR_02	The PISTIS Data Factory Connector shall execute data transfers automatically based on the terms of the smart contract.	UC_01, UC_02, UC_03, UC_04	N/A
FR_03	The PISTIS Data Factory Connector shall provide notifications relevant to the outcome of data transfers.	UC_05, UC_06, UC_07, UC_08	N/A

5.1.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Performance efficiency	NFR1	The overall transaction shall be performed without delays and should not consume unnecessary system resources.
Reliability	NFR2	The PISTIS Data Factory Connector shall operate in a reliable manner, transferring the whole of the data asset that is described in the smart contract and being capable of high resilience.
Reliability	NFR3	The PISTIS Data Factory Connector shall provide notifications to the users.
Security	NFR4	The overall data transfer shall be made through secure communication channels.

5.1.6 Component Architecture

The main elements comprising the PISTIS Data Factory Connector, which handles datasets as files, are:

- The Smart Contract checker which is used to retrieve and analyse the details present in the smart contract to resolve how the system shall proceed with the transfer.
- The Transfer Gateway Registry that is used to store transaction related data locally in order to support certain operations (such as transferring data in batches, etc).
- The Data Factory Storage I/O Service that is retrieving the data stored in the local storage of a PISTIS Data Factory to transfer it. The same component also writes back to the PISTIS Data Factory of the Data Consumer, once he has acquired the data asset.
- The Metadata Repository I/O Service that is retrieving the asset's metadata stored in the PISTIS Data Factory Storage to transfer it. The same component also writes back

the asset's metadata to the Metadata Repository of the Data Consumer, once he has acquired the data asset.

- The Request / Data Reception Service that is used to request a specific dataset (based on an active smart contract) and is also receiving the relevant information (and data asset).
- The Data Publishing Service that is used to bundle the data and the metadata of an asset and send it to the Request / Data Reception Service of the PISTIS Data Factory Connector component that resides at the side of the Data Consumer.

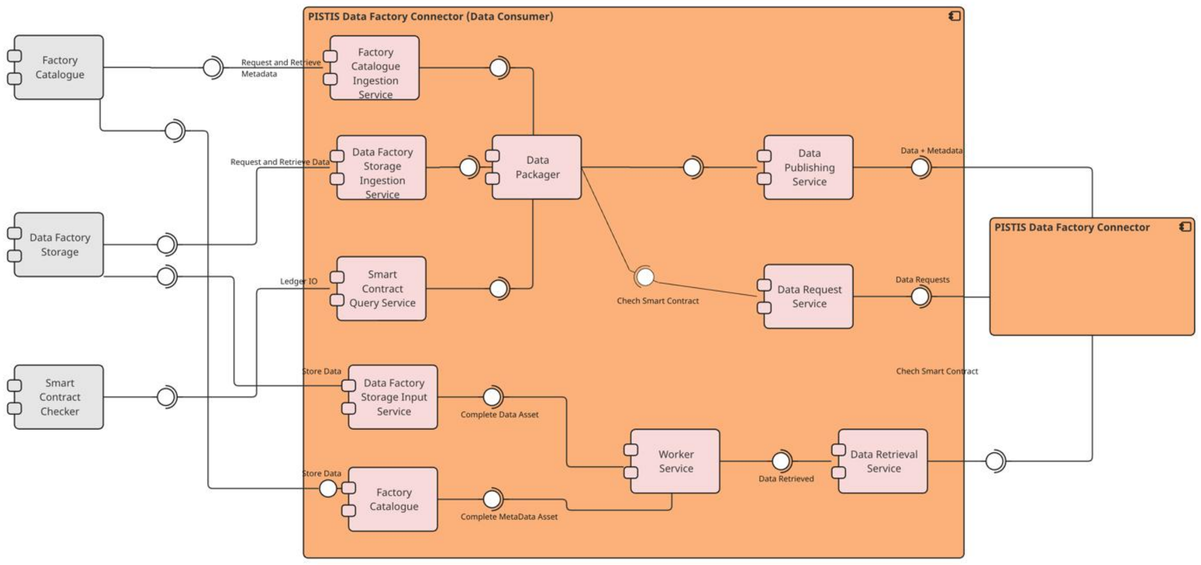


Figure 37: PISTIS Factory Connector's Internal Architecture – Files Transfer

Furthermore, as streaming data is now available to be traded over PISTIS, a new part of the Connector has been developed. Its architecture is shown below.

Once the messages are transferred to the factory Kafka topic, the user who now owns the streaming data, should be able to consume the messages from that topic. To enable this, the Factory UI component requests the Kafka connection details and user credentials from the internal Data Connector module, the Data Consumer Service.

The Kafka service then interacts with the Data Factory Storage component to retrieve the factory details such as the factory name, and with the Kafka broker to obtain the necessary connection information and user credentials.

With the provided connection details and credentials the consumer user can authenticate with Kafka and start consuming messages from the topic. For example, an external application can use this information to connect to the Kafka broker and consume messages, as depicted in the diagram below.

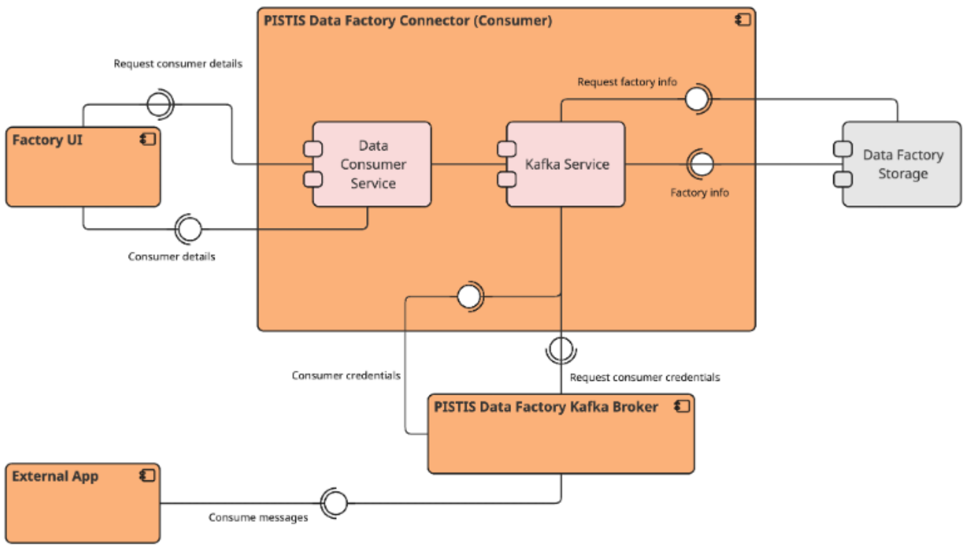


Figure 38: PISTIS Factory Connector's Internal Architecture – Files Transfer

5.1.7 Technology Background

The main technology used for the Component is Node.JS (based on the Nest Framework) as the whole component is a backend service that is deployed at the side of each PISTIS Data Factory and can ingest and provide data to other deployments as instructed by the smart contracts.

5.1.8 Graphical User Interface

N/A - This is a backend service and GUI is not available.

5.1.9 Developer Documentation

The developer guide for this component is available here:

https://docs.pistis-market.eu/developers/connector/connector

5.1.10 Source Code

The source code of this component is available to consortium and reviewers here:

- Backend: https://github.com/PISTIS-Platform/components-monorepo
- Frontend: https://github.com/PISTIS-Platform/cloud-ui and https://github.com/PISTIS-Platform/platform-frontend

5.2 Smart Contract Checker

5.2.1 Component Description

The Smart Contract Checker (SCC) in the PISTIS platform is responsible for making sure every transaction on the network follows the platform's rules. Smart Contract Checker's checking policy is based on two pillars that collectively uphold the security, authenticity, and compliance of operations on the platform:

- First, that every transaction comes from a real, authorized user who has the right credentials and is registered on the platform. This means the SCC verifies all signatures and confirms the user's identity and permission to perform the action.
- Second, it ensures that the transaction itself does not break any rules or restrictions, such as privacy policies, data usage limits, or account balances before the data or asset changes hands.

The SCC acts as a rule-based engine. In the Alpha version, it works with a fixed set of basic rules. These rules are expected to be expanded and updated as the project goes on, allowing the SCC to check new or more complex rules required by the platform operators.

5.2.2 Main Improvements in Beta Version

In the Beta version, the Smart Contract Checker will improve how it applies the official rules for checking transactions. The SCC will perform these checks more strictly and consistently, especially in cases where the Alpha version may not have covered them fully. It will also expand its checks to include both the buyer's and the seller's sides of a transaction. This means that before a transaction happens, the SCC will verify permissions and prohibitions not only for the party selling or sharing data but also for the party buying or receiving it. These improvements will help make sure that all transactions comply with contract and license rules, enhancing the security and trustworthiness of the platform.

5.2.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
US_01	Data Provider	check the data before sharing them (with UBITECH's rules)	I can be sure everything is in the correct form	Alpha	smart contract check functionali ty	Done	PISTIS .OUS. 10, PISTIS .OUS.

							11
US_02	Data Consumer	each data trade I perform to be checked (with UBITECH's rules)	I can be sure of the data validity and correctness	Alpha	smart contract check functionality	Done	PISTIS.OUS.10, PISTIS.OUS.11
US_03	Data Provider, Data Administrator	be sure that before data sharing the necessary amount of money have transferred successfully	I can be sure that the Data Consumer and paid for them	Alpha	smart contract check functionality	Done	PISTIS.OUS.10, PISTIS.OUS.11
US_04	Data Provider, Data Consumer	be sure that before data sharing that data are GDPR compliant	I can be sure that no legal issues will be arise	Alpha	smart contract check functionality	Done	PISTIS.OUS.10, PISTIS.OUS.11
US_05	Data Provider, Data Consumer	be sure that before data sharing that do not violate and policy (with UBITECH's rules)	I can be sure that no issues will be arise	Alpha	smart contract check functionality	Done	PISTIS.OUS.10, PISTIS.OUS.11
US_06	Data Provider	check the data before sharing them (with official rules)	I can be sure everything is in the correct form	Beta	smart contract check functionality	Done	PISTIS.OUS.10, PISTIS.OUS.11
US_07	Data Consumer	each data trade I perform to be checked (with official rules)	I can be sure of the data validity and correctness	Beta	smart contract check functionality	Done	PISTIS.OUS.10, PISTIS.OUS.11
US_08	Data Provider, Data Consumer	be sure that before data sharing that do not violate and policy (with official rules)	I can be sure that no issues will be arise	Beta	smart contract check functionality	Done	PISTIS.OUS.10, PISTIS.OUS.11

5.2.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	Smart Contract Checker supports checking potential violations of a smart contract.	UC_1, UC_2, UC_3, UC_4, UC_5	These could be on the GDPR compliance, on the smart contract business logic, on the monetary values needed for the transaction etc.

5.2.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Performance efficiency	NFR1	Smart Contract check should be performed in efficient way.
Reliability	NFR2	Smart Contract check result should be a reliable report.
Security	NFR3	Only authorised components can trigger smart contract checks.

5.2.6 Component Architecture

The Smart Contract Checker tool is designed to validate the integrity and compliance of smart contracts through a two-step analysis process. It begins with the 'Authenticity Check' module that verifies the originality and correctness of the contract code against predefined standards. Following this, the 'Violations Check' module scans for any breaches of contractual or regulatory rules embedded within the contract logic. The results from these modules are compiled into the 'Smart Contract Checker Result', which details the status of the contract in terms of both authenticity and legal compliance. This systematic approach helps ensure that smart contracts are both genuine and adhere to all applicable laws and regulations.

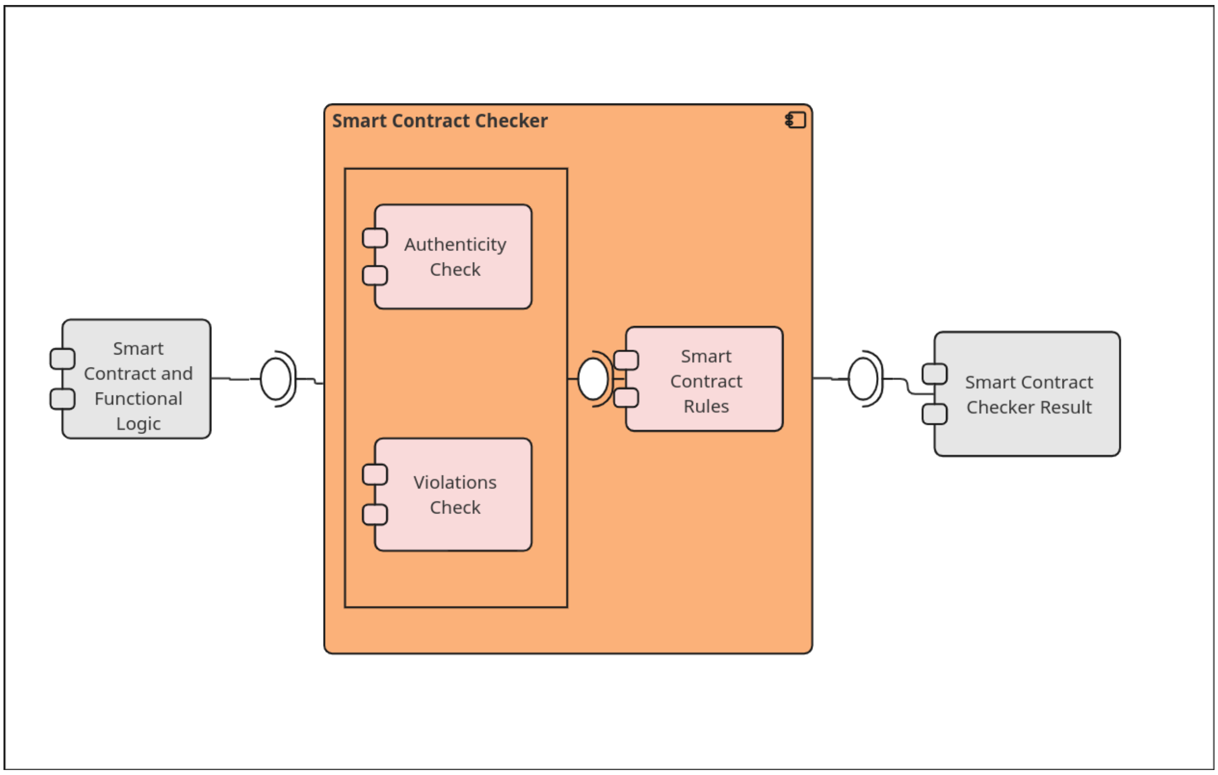


Figure 39: Smart Contract Checker High Level Architecture

5.2.7 Technology Background

The Smart Contract Checker component utilizes modern web technologies to provide compliance solutions in order to filter the provided Smart Contract under a set of rules. At its core, the system is developed using **Node.js** and the application logic and compliance rules are implemented in **TypeScript**.

For its interfacing with other components and external clients, the Smart Contract Checker exposes **RESTful APIs**. These APIs allow for a standardized way of communicating with other parts of the system, facilitating requests for data validation, retrieval of compliance reports, and submission of privacy policies for analysis.

5.2.8 Graphical User Interface

This is a backend component, and therefore no UI is available.

5.2.9 Developer Documentation

The developer documentation for the Smart Contract Checker component may be accessed on the official PISTIS project documentation site at the following link: https://docs.pistis-market.eu/developers/smart-contract-checker/smart-contract-checker

5.2.10 SOURCE CODE

The source code of the Smart Contract Checker (SCC) component is maintained in a private GitHub repository under the PISTIS project, accessible only to the PISTIS consortium and the reviewers at

https://github.com/PISTIS-Platform/besu_ledger-scee-scc-gdpr-checker/tree/main/node-server/src/checkers/smartContractChecker

While the SCC is conceptually distinct, it is technically integrated within the same Node.js server as the Smart Contract Execution Engine (SCEE) and the GDPR Checker, all hosted in the same repository. Additionally, the repository includes the Data Ledger implementation, developed using the Hyperledger Besu Quorum framework, providing a unified solution for these interconnected components. The attached link leads to the SCC's implementation logic within this comprehensive repository.

6 AI & Interoperability Repositories Bundle

The AI & Interoperability Repos bundle provides the different repositories for storing and propagating different models (data models, AI models and metadata models) that need to be consumed by the different components.

This bundle consists of the following components:

- PISTIS Models Repository
- Data Factory ML Models Repository
- AI Model Editor

These are presented in the following sub-sections.

6.1 PISTIS Models Repository

6.1.1 Component Description

The PISTIS Models Repository is responsible for the storage, management, and governance of all models that are to be used by the different PISTIS components.

These models include:

- Data models that define entities, attributes, and relationships within a specific domain. Data providers must use the data models when describing their actual data and a browser for these models will be available.
- Metadata models that define the metadata that shall be provided to accompany each dataset traded over PISTIS. This repository is s based on the RDF standard and established sub-standards, such as DCAT and Gaia-X self-descriptions. The Data Catalogue component reads the metadata models during its start-up process to configure itselves accordingly. Additionally, the Metadata Model Management ensures the automatic generation of a machine-and-human-readable documentation.
- Monetisation AI models, which are used by the analytics engine residing in the Cloud platform to accommodate the needs of the PISTIS Market Insights component.

The PISTIS Models Repository also supports version control for the PISTIS models, allowing users to track changes over time, which is crucial for managing updates and ensuring consistency across the whole PISTIS ecosystem. It also maintains metadata associated with each data model, providing information about its version, size, and creation/last update date. Users can search and retrieve specific data models based on various criteria, facilitating easy access to relevant information.

The PISTIS Platform administrator is the role that maintains the models and can upload new artefacts and fill-in new metadata information on the existing ones, as well as to remove specific models.

6.1.2 Main Improvements in Beta Version

In this version, the PISTIS Models Repository also offers to the PISTIS Platform Administrators the option to download or delete a model from the repository.

6.1.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	PISTIS Platform administrator	view all the content available in the PISTIS models repo	I am aware of the available models and artefacts	Alpha	Display all contents (models, artefacts) of the repo	Done	PISTIS. SOUS. 02
UC_02	PISTIS Platform administrator	upload an artefact in the repo	it can become available to the other components	Alpha	Uploaded artefact visible in the PISTIS Repository	Done	PISTIS. SOUS. 02
UC_03	PISTIS Platform administrator	edit the description and metadata of an artefact in the repo	other viewers can understand more about it.	Alpha	Display new model description and metadata	Done	PISTIS. SOUS. 02
UC_04	Data Factory Component	get a view of the repositories contents via API	I can see what is inside	Alpha	APIs available	Done	PISTIS. SOUS. 02
UC_05	Data Factory Component	be able to get a specific model	I can use it internally	Alpha	Model File downloaded/ saved	Done	PISTIS. SOUS. 02
UC_06	PISTIS Platform administrator	add a new version of a model	I keep version	Alpha	New version of the model added	Done	PISTIS. SOUS. 02
UC_07	PISTIS Platform administrator	select an artefact of the repo	I can download it	Beta	artefact downloaded/ saved	Done	PISTIS. SOUS. 02
UC_08	PISTIS Platform administrator	select multiple artefacts of the repo	I can download them	V1.00	Selected artefact downloaded/ saved	Upcom ing	PISTIS. SOUS. 02
UC_09	PISTIS Platform administrator	select an artefact of the repo	I can delete it	Beta	Selected artefact deleted	Done	PISTIS. SOUS. 02
UC_10	PISTIS Platform administrator	select multiple artefacts of the repo	I can delete them	V1.00	Selected artefacts deleted	Upcom ing	PISTIS. SOUS. 02
UC_11	PISTIS Platform administrator	Remove selected artefacts of the repo	They are no longer part of the repository	V1.00	Selected artefacts deleted	Upcom ing	PISTIS. SOUS. 02

6.1.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	The PISTIS Models Repository shall provide a user-friendly interface allowing the Platform administrator to view/edit/download/delete/manage the PISTIS data models.	US_01, US_02, US_03, US_07, US_08, US_09, US_10, US_11	
FR_02	The PISTIS Models Repository shall enable the model Administrator to create a new model and populate it along with its metadata.	US_02, US_03	
FR_03	The PISTIS Models Repository shall enable the Platform administrator to edit the metadata of an existing model.	US_03	
FR_03	The PISTIS Models Repository shall enable the Platform administrator to upload a new version of an existing model.	US_06	

6.1.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Performance efficiency	NFR1	The overall process shall be performed without delays and should not consume unnecessary system resources.
Reliability	NFR2	The component shall operate in a reliable manner, providing reliable information to the PISTIS Model Manager.
Security	NFR3	The overall process shall be made through secure communication channels.

6.1.6 Component Architecture

The PISTIS Models Repository consists of the following components:

- Frontend Repository Management Service: The user interface (dashboard) that enables users (Platform Administrators) to take actions, related to upload new data artefact, edit the existing models, etc.
- PISTIS Model Manager Backend: This component executes the user's actions (upload, edit, delete), communicating with the Global Model Storage towards retrieving and/or updating the available models, uploading new ones, etc.
- Data Models Repository: Refers to the actual storage facility where the models (as files) are residing.
- Repository Exposure API: The API gateway used by the other component to read the models that are stored in the repository.

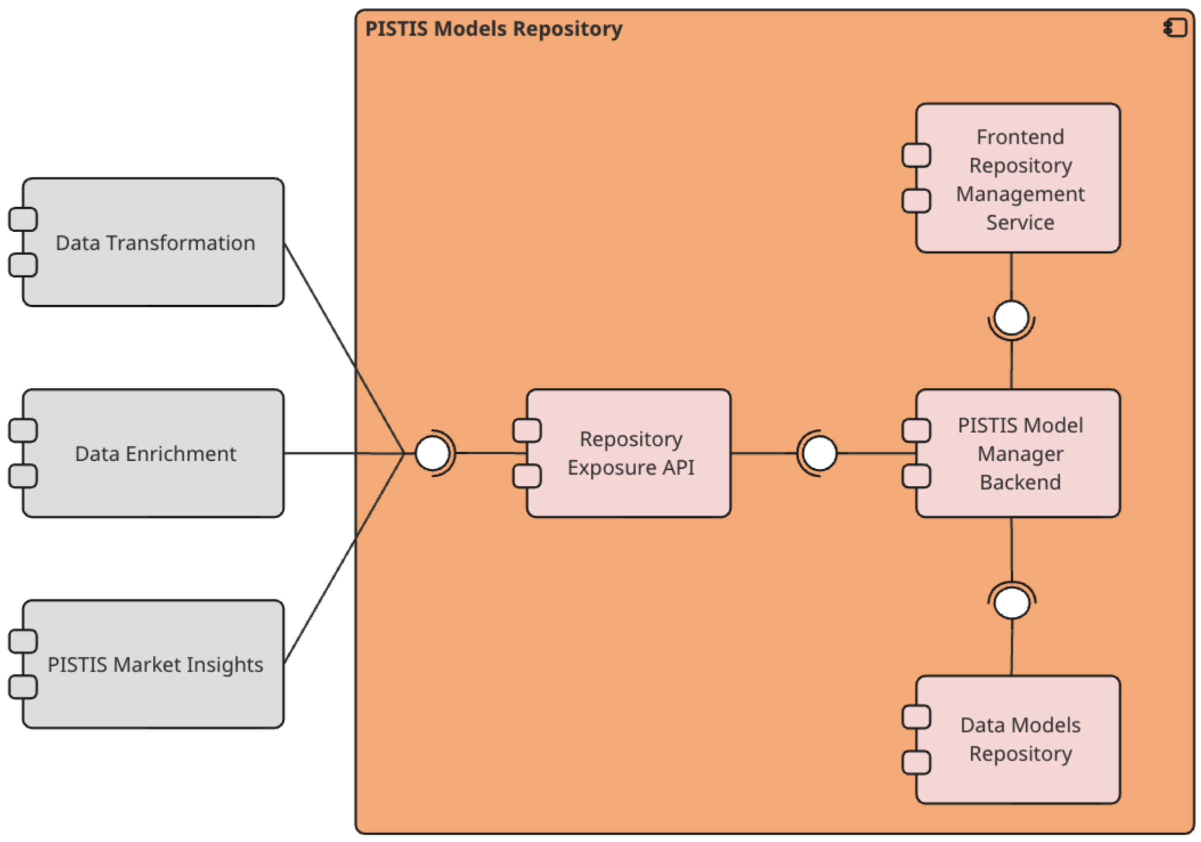


Figure 40: PISTIS Model Repository Internal Architecture

6.1.7 Technology Background

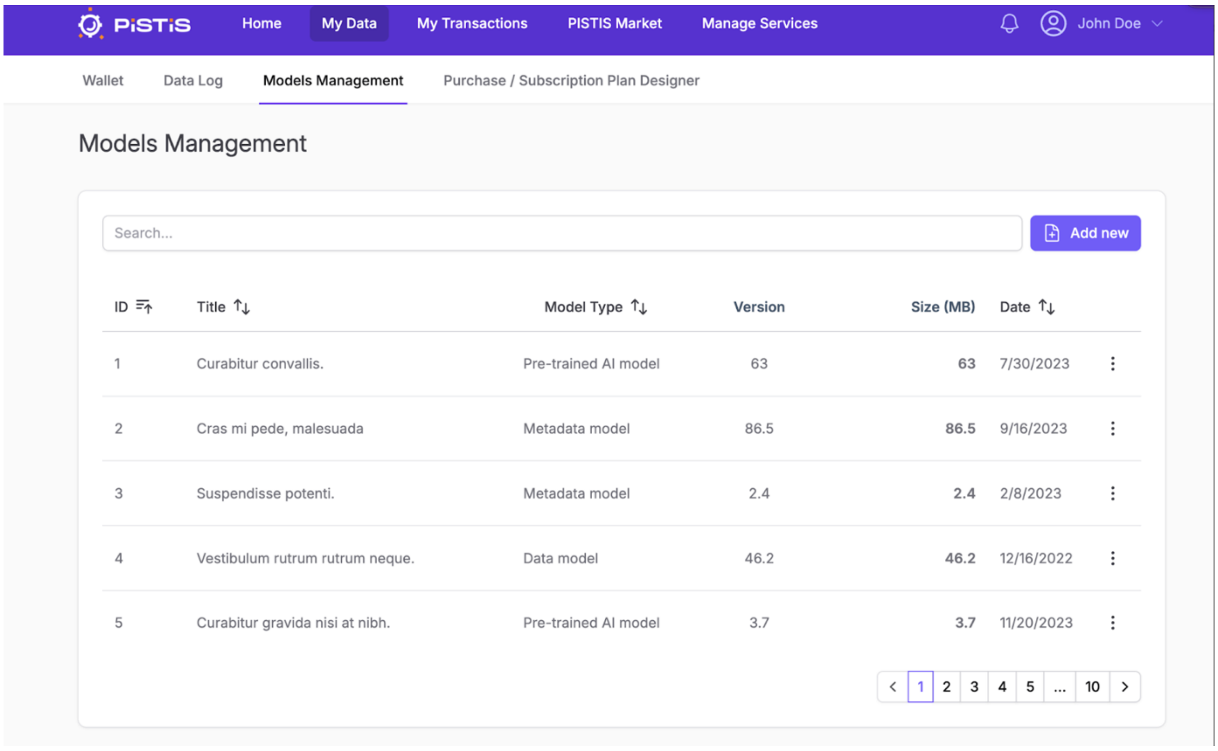
The PISTIS Models Repository will offer a frontend service, developed in Nuxt.js and Vue.js, delivering the UI where the PISTIS data model Administrator will be able to generate, upload edit and manage the PISTIS data models.

For the backend services of this component and regarding the data, and the Monetisation AI models, Nest.js will be exploited, while intra-component communication will be facilitated with Rest APIs.

For the Metadata Model repository, the technology stack to be used includes SHACL, OWL, SKOS, DCAT-AP for Data Spaces, as well as Custom SHACL Extensions.

6.1.8 Graphical User Interface

The PISTIS Models Repository enables the PISTIS Administrator to view the list of the models residing in the repository as shown below.

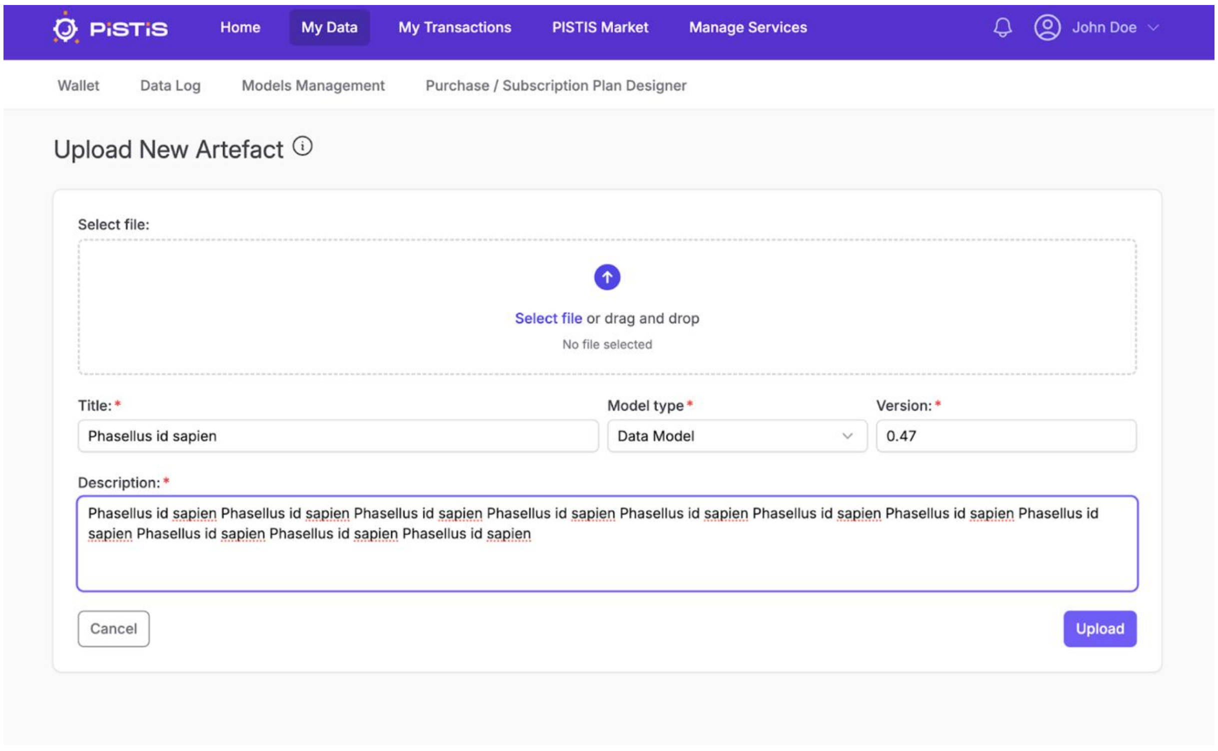


Models Management

ID	Title	Model Type	Version	Size (MB)	Date
1	Curabitur convallis.	Pre-trained AI model	63	63	7/30/2023
2	Cras mi pede, malesuada	Metadata model	86.5	86.5	9/16/2023
3	Suspendisse potenti.	Metadata model	2.4	2.4	2/8/2023
4	Vestibulum rutrum rutrum neque.	Data model	46.2	46.2	12/16/2022
5	Curabitur gravida nisi at nibh.	Pre-trained AI model	3.7	3.7	11/20/2023

Figure 41: PISTIS Models Repository – Models Management

The PISTIS Administrator has the option to upload a new artefact through the interface depicted below.



Upload New Artefact

Select file:

Title: * Phasellus id sapien

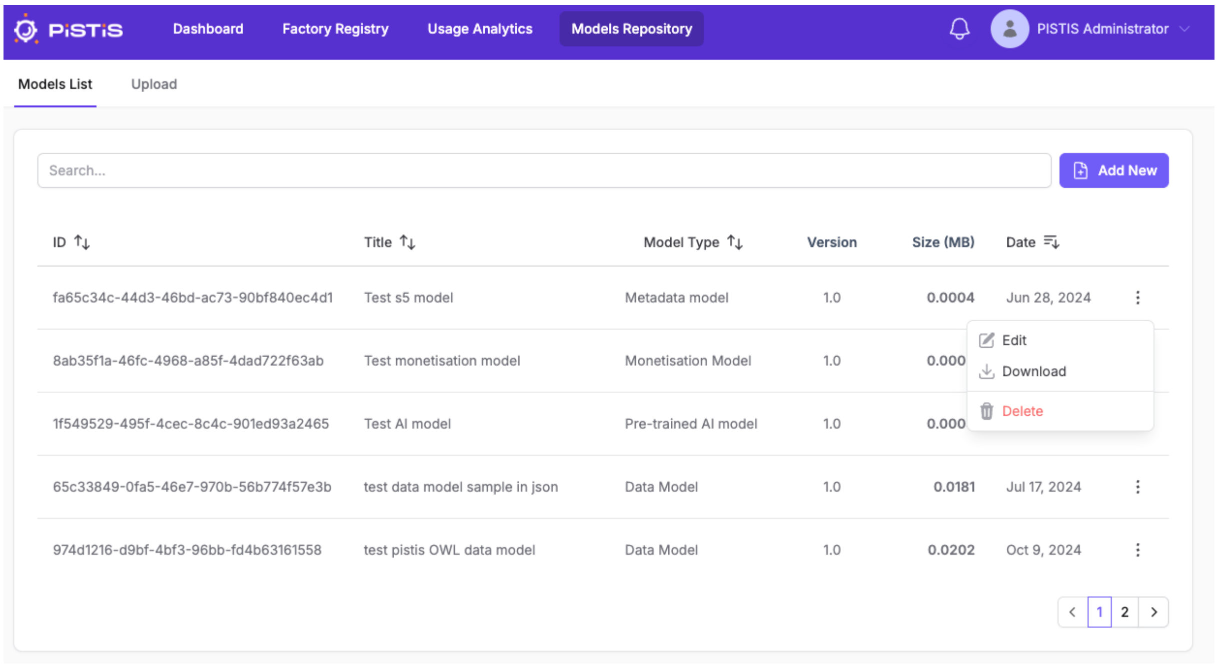
Model type * Data Model

Version: * 0.47

Description: * Phasellus id sapien Phasellus id sapien Phasellus id sapien Phasellus id sapien Phasellus id sapien Phasellus id sapien Phasellus id sapien Phasellus id sapien Phasellus id sapien Phasellus id sapien Phasellus id sapien

Figure 42: PISTIS Models Repository – Upload of New Artefact

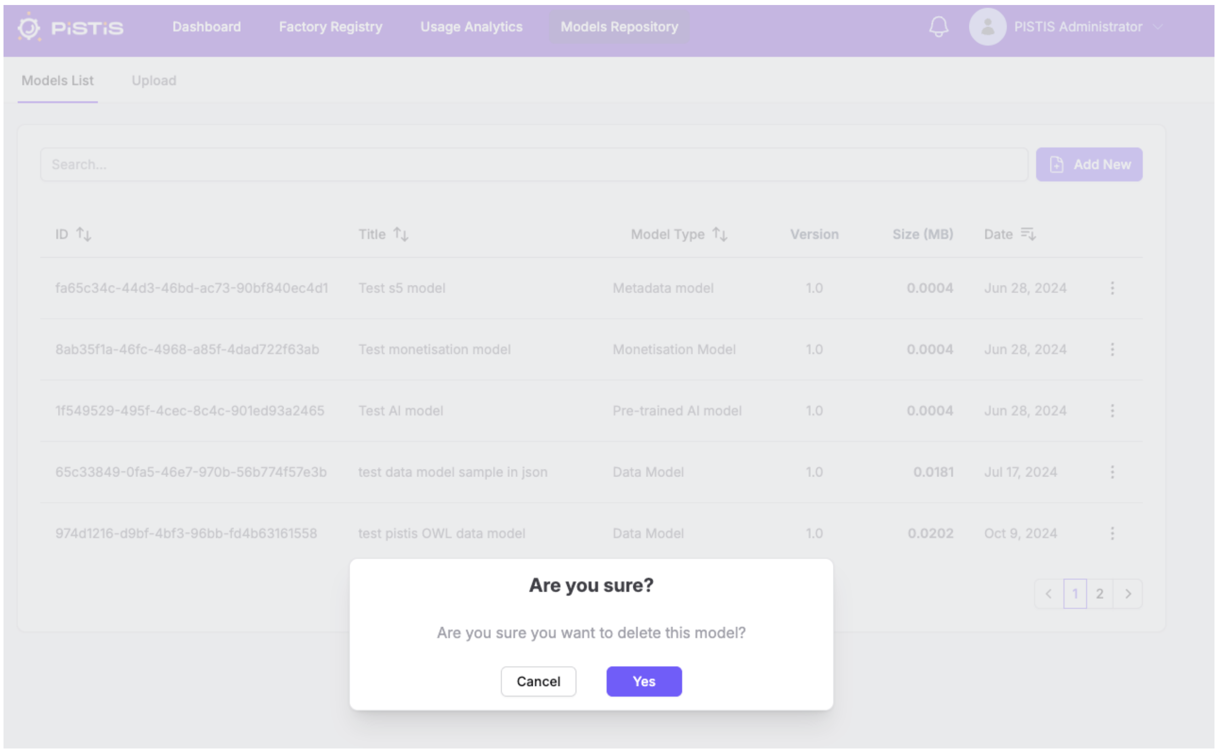
The PISTIS Administrator has the option to download or delete a model as shown below.



ID	Title	Model Type	Version	Size (MB)	Date
fa65c34c-44d3-46bd-ac73-90bf840ec4d1	Test s5 model	Metadata model	1.0	0.0004	Jun 28, 2024
8ab35f1a-46fc-4968-a85f-4dad722f63ab	Test monetisation model	Monetisation Model	1.0	0.000	
1f549529-495f-4cec-8c4c-901ed93a2465	Test AI model	Pre-trained AI model	1.0	0.000	
65c33849-0fa5-46e7-970b-56b774f57e3b	test data model sample in json	Data Model	1.0	0.0181	Jul 17, 2024
974d1216-d9bf-4bf3-96bb-fd4b63161558	test pistis OWL data model	Data Model	1.0	0.0202	Oct 9, 2024

Figure 43: PISTIS Models Repository – Download/Delete an Artefact

When the PISTIS Administrator deletes a model, a popup message appears to confirm the deletion.



ID	Title	Model Type	Version	Size (MB)	Date
fa65c34c-44d3-46bd-ac73-90bf840ec4d1	Test s5 model	Metadata model	1.0	0.0004	Jun 28, 2024
8ab35f1a-46fc-4968-a85f-4dad722f63ab	Test monetisation model	Monetisation Model	1.0	0.0004	Jun 28, 2024
1f549529-495f-4cec-8c4c-901ed93a2465	Test AI model	Pre-trained AI model	1.0	0.0004	Jun 28, 2024
65c33849-0fa5-46e7-970b-56b774f57e3b	test data model sample in json	Data Model	1.0	0.0181	Jul 17, 2024
974d1216-d9bf-4bf3-96bb-fd4b63161558	test pistis OWL data model	Data Model	1.0	0.0202	Oct 9, 2024

Are you sure?

Are you sure you want to delete this model?

Figure 44: PISTIS Models Repository – Delete confirmation

6.1.9 Developer Documentation

The developer guide for this component is available here:

https://docs.pistis-market.eu/developers/models-repository/models-repository

6.1.10 Source Code

The source code of this component is available to consortium and reviewers here:

- Backend: https://github.com/PISTIS-Platform/components-monorepo
- Frontend: https://github.com/PISTIS-Platform/cloud-ui

6.2 Data Factory ML Models Repository

6.2.1 Component Description

The Data Factory ML Models repository will provide support for CRUD and serving operations over a concrete pre-trained model.

This repository is essentially a similar deployment of the PISTIS Models Repository, but concerns only ML models which can be uploaded by Data Factory users to run analyses over their own data, while they can also fetch already Pre-trained ML models from the PISTIS Models Repository

6.2.2 Main Improvements in Beta Version

The following improvements have been introduced in beta version:

- Fetch a Model from the PISTIS Models Repository
- Serve a Model

6.2.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	User	Add a Model	Add a Model	Alpha	Model added	Done	PISTIS.OUS.03

UC_02	User	Fetch a Model from the PISTIS Models Repository	Keep Model updated	Beta	Model descriptions updated	Done	PISTIS.OUS.03
UC_03	User	Serve a Model	Enabling trained models are made available for others to use	Beta	Model served	Done	PISTIS.OUS.03
UC_04	User	Add or Updating Model Descriptions	Update descriptions over the model	Alpha	Model updated	Done	PISTIS.OUS.03
UC_05	User	Edit a Model's Metadata	Support new naming	Alpha	Model renamed	Done	PISTIS.OUS.03
UC_06	User	List and searching Models	Found Models	Alpha	Check searching	Done	PISTIS.OUS.03
UC_07	User	Archive a Model	Archive Model	Alpha	Model archived	Done	PISTIS.OUS.03
UC_08	User	Delete Models	Delete Models	Alpha	Model deleted	Done	PISTIS.OUS.03

6.2.4 Functional Requirements

This section provides the functional requirements of the ML Model Repository component:

ID	Description	Related Use Cases	Comments
FR_01	Manage storing of Pre-trained AI Models.	UC1, UC2, UC4, UC5, UC6, UC7, UC8	
FR_02	Serve Pre-trained AI Models.	UC3	

6.2.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Security	NFR1	PISTIS ensures that only authorised user can register datasets.

6.2.6 Component Architecture

As it is shown in Figure 45, the architecture of the ML model repository is built mostly on the use of MinIO as the primary technology for ML model repository. The Rest API merely exposes a collection of functions related to an ML model's life cycle and, via an internal proxy, connects to the MinIO backend.

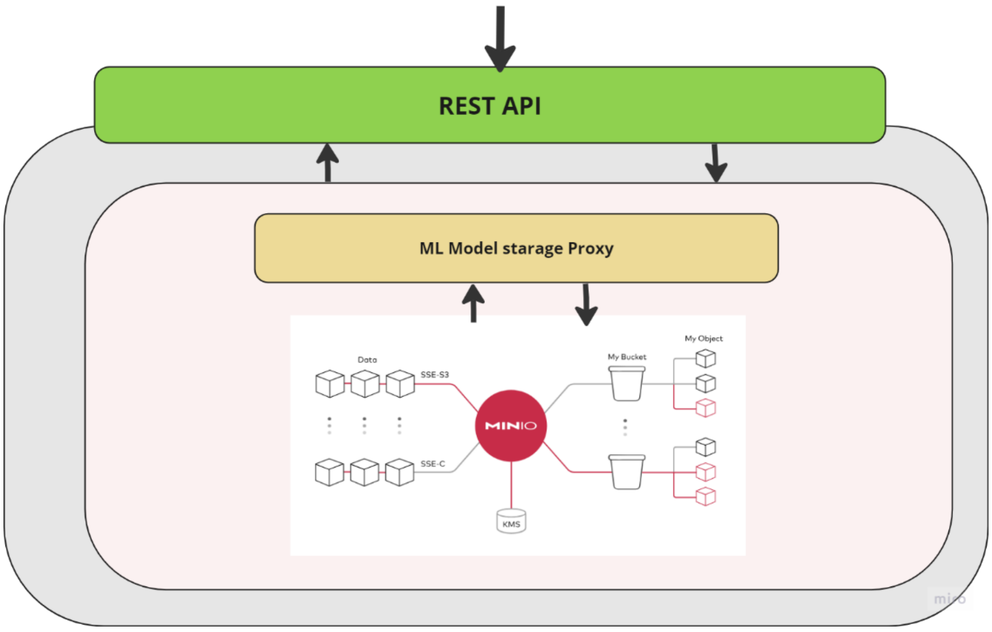


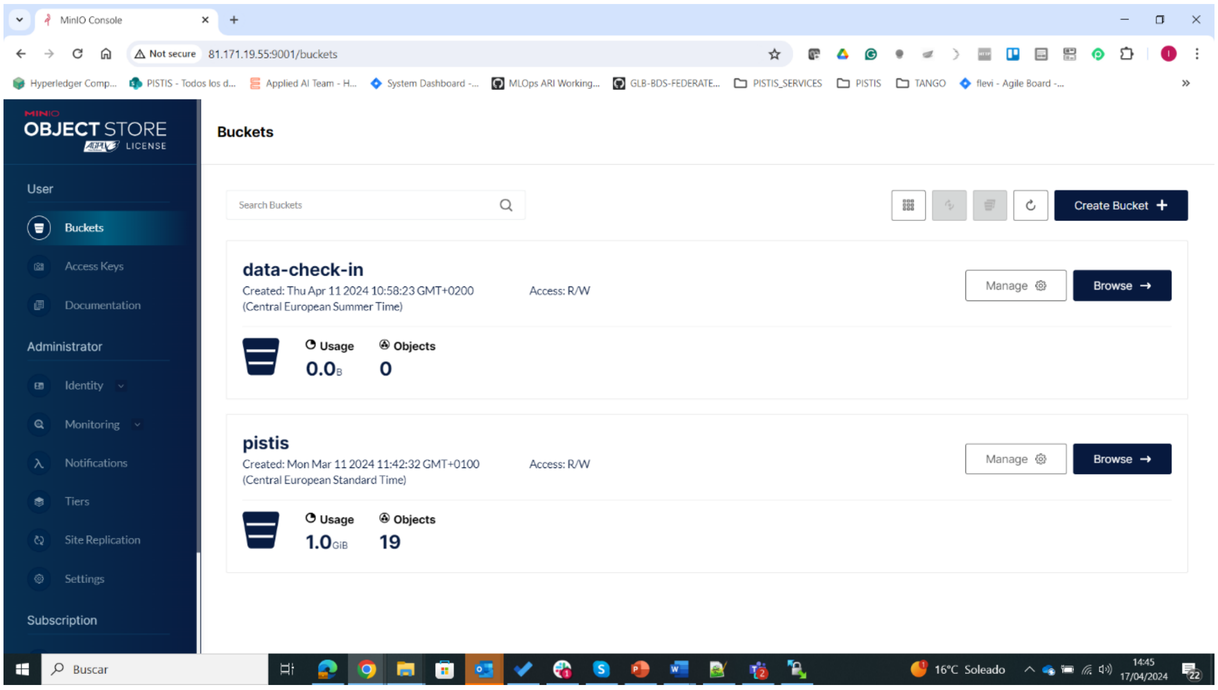
Figure 45: ML Model Repo Architecture

6.2.7 Technology Background

The main technology used for the Component is **MinIO** integrated with MLFLow.

6.2.8 Graphical User Interface

The ML Model Repository's UI will not involve GUI development. The MinIO Console is intended to be used as a result of the MinIO technology stack being used. Consequently, MinIO Console will serve as the GUI for our component, assisting with administration tasks such as Identity and Access Management, Metrics and Log Monitoring, and server configuration. The MinIO Console is incorporated in the MinIO Server, which is part of the ML Model Repository component. A screenshot of MinIO Console is shown in Figure 46.



Buckets

data-check-in
Created: Thu Apr 11 2024 10:58:23 GMT+0200 (Central European Summer Time)
Access: R/W
Usage 0.0B
Objects 0

pistis
Created: Mon Mar 11 2024 11:42:32 GMT+0100 (Central European Standard Time)
Access: R/W
Usage 1.0GiB
Objects 19

Figure 46: MinIO GUI

6.2.9 Developer Documentation

Because the ML Model Repository component is built on MLFlow/MinIO integration, which is a third-party technology, the component's documentation is publicly available at https://mlflow.org/docs/1.30.0/index.html for MLFlow and https://min.io/docs/kes for MinIO.

6.2.10 Source Code

The ML Model Repository component is based on 3rd party technology called MinIO whose open-source repository is accessible at https://github.com/minio/minio.

6.3 AI Model Editor

6.3.1 Component Description

The AI Model Editor will provide support for creating, editing, and sharing computational AI models.

AI Model Editor will allow the end user to cover a basic workflow that includes at least the following tasks:

- Create a project enabling collaboration (or not) with others to work with data.
- Add a notebook to the project.
- Add code and run the notebook.
- Review the model pipelines and save the desired pipeline as a model.
- Deploy and test a concrete model.

6.3.2 Main Improvements in Beta Version

The following improvements have been introduced in beta version:

- Deploy and test a concrete model.
- Fetch a Model from the PISTIS Models Repository.

6.3.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data Consumer	Create a project with the AI editor	Manage AI Models	Alpha	Check project created properly	Done	PISTIS.OUS.03, PISTIS.OUS.07
UC_02	Data Consumer	Add a notebook to project	Manage AI Models	Alpha	Notebook added properly	Done	PISTIS.OUS.03, PISTIS.OUS.07
UC_03	Data Consumer	Add code and run the notebook	Manage AI Models	Alpha	Check the notebook code	Done	PISTIS.OUS.03, PISTIS.OUS.07
UC_04	Data Consumer	Review the model pipelines and save the desired pipeline as a model	Manage AI Models	Alpha	Check that model saved exists	Done	PISTIS.OUS.03, PISTIS.OUS.07
UC_05	Data Consumer	Deploy and test a concrete model	Deploy and test AI Models	Beta	Model accessible to be tested.	Done	PISTIS.OUS.03, PISTIS.OUS.07

6.3.4 Functional Requirements

This section provides the functional requirements of the ML Model Editor:

ID	Description	Related Use Cases	Comments
FR_01	Create, edit, and sharing computational AI models.	UC1, UC2, UC3, UC4 and UC5	

6.3.5 Non-Functional Requirements

There are no non-functional requirements at the moment.

6.3.6 Component Architecture

Regarding the ML Model Editor component, reusing current open-source technologies is the preferred approach over developing the component itself. Since Jupyter Lab was the technology chosen in this instance, Figure 47, which displays the many architectural layouts of the essential components of the Jupyter ecosystem, can be used as a point of reference.

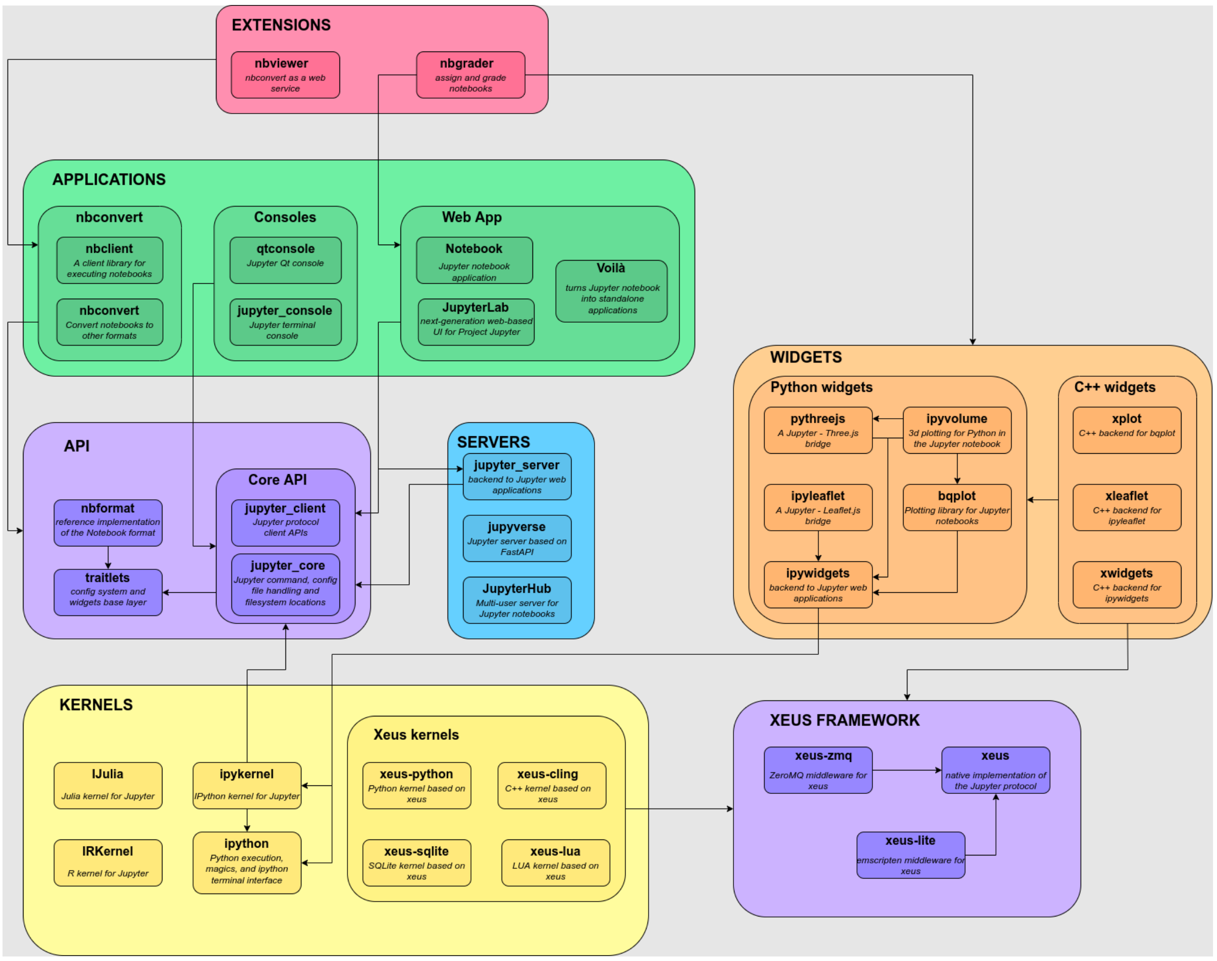

EXTENSIONS
nbviewer
nbconvert as a web service
nbgrader
assign and grade notebooks
APPLICATIONS
nbconvert
nbclient
A client library for executing notebooks
nbconvert
Convert notebooks to other formats
Consoles
qtconsole
Jupyter Qt console
jupyter_console
Jupyter terminal console
Web App
Notebook
Jupyter notebook application
JupyterLab
next-generation web-based UI for Project Jupyter
Voilà
turns Jupyter notebook into standalone applications
API
nbformat
reference implementation of the Notebook format
traitlets
config system and widgets base layer
Core API
jupyter_client
Jupyter protocol client APIs
jupyter_core
Jupyter command, config file handling and filesystem locations
SERVERS
jupyter_server
backend to Jupyter web applications
jupyverse
Jupyter server based on FastAPI
JupyterHub
Multi-user server for Jupyter notebooks
WIDGETS
Python widgets
pythreejs
A Jupyter - Three.js bridge
ipyvolume
3d plotting for Python in the Jupyter notebook
ipyleaflet
A Jupyter - Leaflet.js bridge
bqplot
Plotting library for Jupyter notebooks
ipywidgets
backend to Jupyter web applications
C++ widgets
xplot
C++ backend for bqplot
xleaflet
C++ backend for ipyleaflet
xwidgets
C++ backend for ipywidgets
KERNELS
IJulia
Julia kernel for Jupyter
IRKernel
R kernel for Jupyter
ipykernel
IPython kernel for Jupyter
ipython
Python execution, magics, and ipython terminal interface
Xeus kernels
xeus-python
Python kernel based on xeus
xeus-cling
C++ kernel based on xeus
xeus-sqlite
SQLite kernel based on xeus
xeus-lua
LUA kernel based on xeus
XEUS FRAMEWORK
xeus-zmq
ZeroMQ middleware for xeus
xeus
native implementation of the Jupyter protocol
xeus-lite
emscripten middleware for xeus

Figure 47: AI Model Editor Architecture[10]

[10] https://docs.jupyter.org/en/latest/projects/architecture/content-architecture.html

6.3.7 TECHNOLOGY BACKGROUND

The main technology used for the Component is **Jupyter Notebook**.

6.3.8 GRAPHICAL USER INTERFACE

In the case of the ML Model Editor component, the development of a UI itself is not contemplated but rather the use of GUI associated with the technology used for said component, which is Jupyter Lab.

The Jupyter Lab interface consists of a primary work area with tabs for documents and activities, a collapsible left sidebar, and a menu bar. The left sidebar includes a file browser, a list of running kernels and terminals, the command palette, the notebook cell tools inspector, and a list of tabs.

A screenshot of Jupyter Lab UI is shown in Figure 48.

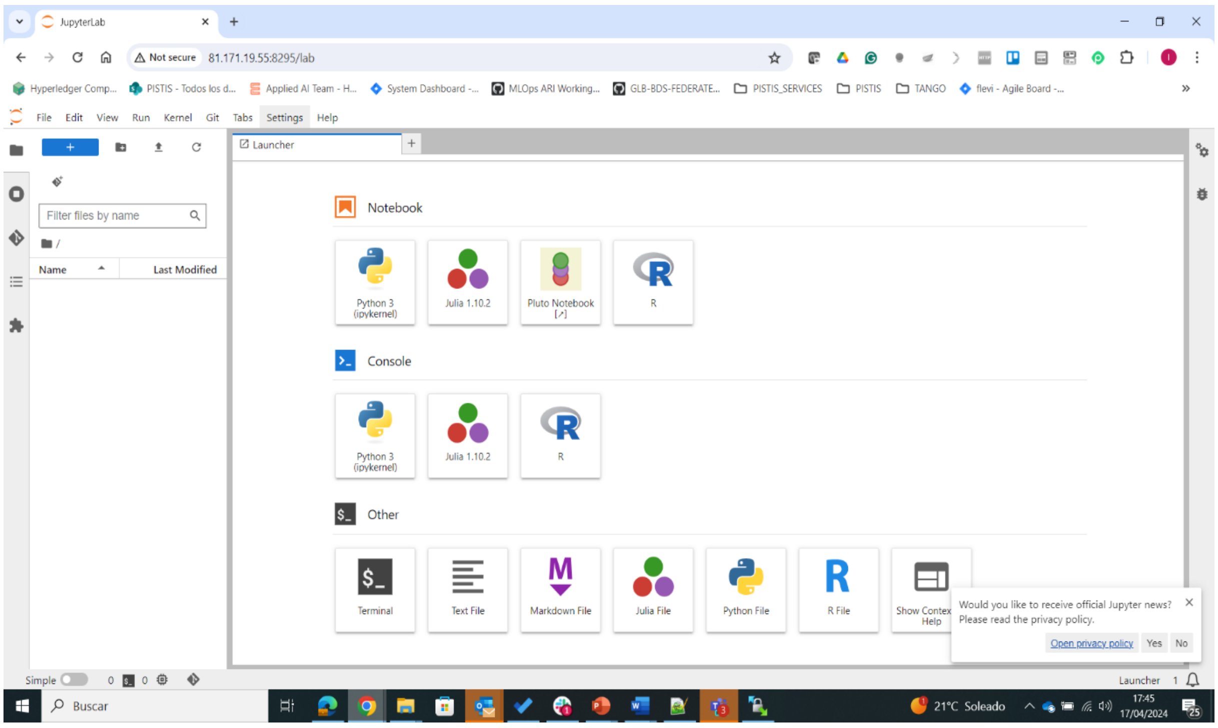


Figure 48: Jupyter Lab UI.

6.3.9 DEVELOPER DOCUMENTATION

The documentation for the ML Model Editor component is publicly available at https://jupyterlab.readthedocs.io/en/latest since it is based on the third-party technology Jupyter Lab.

6.3.10 Source Code

The ML Model Editor component is based on 3rd party technology called Jupyter-Lab whose open-source repository is accessible at https://github.com/jupyterlab/jupyterlab.

7 Security, Trust & Privacy Preservation Bundle

The Security, Trust & Privacy Preservation bundle offers services for strengthening data security and privacy.

This bundle consists of the following components:

- Anonymizer
- Lineage Tracker
- GDPR checker
- Searchable Encryption
- Encryption/Decryption Engine
- Access Policy Editor

These are presented in the following sub-sections.

7.1 Anonymizer

7.1.1 Component Description

The anonymiser is a component responsible for preserving data privacy. It alters data in such a way that it will preserve its usefulness but hides the original data. With these modifications, it cannot be traced back to the individuals the data was taken from.

The anonymiser is capable of taking a dataset and obfuscating the contained data by replacing it with values that represent the original data in a way that is non-identifying (e.g. an age of 29 may be replaced with [20-30] or a name Darren Smith may be replaced with Darren *****). This is known as data masking.

Via the frontend interface, users will be able to configure the anonymisation process by selecting different anonymisation pre-sets, which can be applied to a column in their dataset, or they can use advanced settings to allow for more configurability in their anonymisation. To see how their choices will impact the result, a preview button is available. Upon clicking this button, users will see a subset of their dataset with their current anonymisation options applied to it. This will help users understand the impact of their choices.

The PseudoID generator is a smaller component, capable of producing a unique ID for a user who wishes to share their data as an anonymous user. This ID may then be used for the purposes of communication with the data provider whilst preserving their anonymity.

The Anonymizer also supports ‘Location Privacy’. ‘Location Privacy’ is defined as “the ability of an individual to move in public space with the expectation that under normal circumstances their location will not be systematically and secretly recorded for later use”.

The existence of location databases stripped of identifying tags can leak information. For instance, if I know that Vera is the only person who lives on Dead End Lane, the datum that someone used a location-based service on Dead End Lane can be reasonably linked to Vera.

Since location privacy definition and requirements differ depending on the scenario, no single technique is able to address the requirements of all location privacy categories. Therefore, in the past, the research community, focusing on providing solutions for the protection of location privacy of users, has defined techniques that can be divided into three main classes: *anonymity-based, obfuscation-based*, and *policy-based* techniques. These classes of techniques are partially overlapped in scope and could be potentially suitable to cover requirements coming from one or more of the categories of location privacy.

It is easy to see that anonymity-based and obfuscation-based techniques can be considered dual categories. Anonymity-based techniques have been primarily defined to protect identity privacy and are not suitable for protecting position privacy, whereas obfuscation-based techniques are well suited for position protection and not appropriate for identity protection. Anonymity-based and obfuscation-based techniques could also be both exploited for protecting path privacy. Policy-based techniques are in general suitable for all location privacy categories, although they are often difficult to understand and manage for end users.

It is important to consider the notion of utility within the context of anonymising location data – if the data seeker is looking to understand different groups mobility patterns to inform public transport planning for example accurate location data over time at scale is imperative. Therefore, supporting location privacy has to also consider the impact on the utility of that data – it will impact the monetary value of that data if the necessary insights can no longer be reliably derived from that data.

Location Privacy as a result is supported through a mechanism to generate new **Synthetic Data** that has the **same format and statistical properties** as the original location data.

Synthetic data can then be used to supplement, augment and in some cases replace real data when training Machine Learning models. Additionally, it enables the testing of Machine Learning or other data dependent software systems without the risk of exposure that comes with data disclosure.

7.1.2 Main Improvements in Beta Version

Synthetic Data Generator

The synthetic data generator is a module that produces artificial datasets designed to replicate the statistical properties and structural patterns of real-world data. It uses advanced techniques, powered by deep learning, to create data that is not linked to any actual individuals but still mirrors the complexity and variability of authentic datasets. This approach is particularly valuable for overcoming challenges related to data scarcity, privacy, and regulatory constraints, allowing organizations to generate large-scale, customised datasets on demand for analytics, software testing, and machine learning applications. Because synthetic data is not derived from real individuals, it can be freely shared and used in environments where the use of genuine personal data would be restricted or require significant compliance efforts.

The benefits of synthetic data generation go beyond privacy. Synthetic data can be tailored to address specific needs, such as balancing underrepresented classes in a dataset or removing biases present in the original data. This flexibility enables data scientists and organizations to accelerate development cycles, reduce costs, and enhance the performance of AI models by providing more diverse and representative training data. Furthermore, synthetic data serves as a safe and fully anonymous alternative to real data, making it a powerful tool for industries with strict privacy requirements, such as healthcare, finance, and telecommunications

Risk of Reidentification

The risk of re-identification module refers to the possibility that anonymised or de-identified data could be matched with external information to reveal the identity of individuals whose data was meant to remain confidential. This risk arises because even after direct identifiers (like names or social security numbers) are removed, unique combinations of quasi-identifiers (such as age, gender, and ZIP code) can still make it possible to single out individuals, especially when auxiliary datasets are available for cross-referencing. Advances in data analytics and the proliferation of publicly available data have made re-identification increasingly feasible, challenging traditional anonymisation methods and raising concerns about the true effectiveness of data de-identification.

The consequences of re-identification are significant for both individuals and organizations. If re-identification occurs, it can lead to privacy violations, regulatory penalties under laws like GDPR and HIPAA, and a loss of trust among customers or patients. Organizations may also suffer reputational damage and face public backlash if they are found to have inadequately protected sensitive information. As a result, robust risk assessment and mitigation strategies are essential components of any data anonymisation process, ensuring that the likelihood of re-identification is minimized before data is shared or analysed.

7.1.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status
	As a <Role>	I want to <Action>,	so that <Reason>			
UC_01	Data Provider	apply anonymisation to my data	personal information can be hidden	Alpha	Data Provider is able to hide Personally Identifiable Information (PII) from the Data Seeker	Done
UC_02	Data Provider	create a fake ID	to hide my real ID that my data belongs to	Alpha	Data Provider is able to generate a Pseudo Identity to hide their real identity	Done
UC_03	Data Provider	be able to select a pre-set anonymisation level	I can anonymise data using a specific approach	Alpha	Data Provider is able to select the anonymisatio n approach they wish to apply to the dataset	Done
UC_04	Data Provider	Be able to have more granular control over the anonymisation configuration	I have flexibility over how my data is shared	Alpha	Data Provider is able to control the settings within an anonymisatio n pre-set	Done
UC_05	Data Provider	Generate Synthetic Data with same properties as my data	It is harder to attribute the data back to me	Beta	Data Provider is able to create an equivalent synthetic dataset that has the same mathematical and statistical properties as the original real dataset	Done

7.1.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	Allow the user to select what data columns they want to anonymise.	PISTIS.OUS.03	

FR_02	Allow the user to customise what anonymisation level they wish to apply to their data.	PISTIS.OUS.03	
FR_03	Present a preview of the result of the users selected anonymisation configuration.	PISTIS.OUS.03	
FR_04	Remove Personally Identifiable Information (PII) or other sensitive data.	PISTIS.OUS.03	
FR_05	Obfuscate PII or other sensitive data.	PISTIS.OUS.03	
FR_06	Obfuscate Location Data	PISTIS.OUS.03	
FR_07	Data Swapping	PISTIS.OUS.03	
FR_08	Inject random statistical noise into statistical and machine-learning analyses carried out over the data.	PISTIS.OUS.03	
FR_09	Present a preview of an anonymised dataset.	PISTIS.OUS.03	

7.1.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Functional Suitability	NFR1	effectively anonymises user data, obscuring identifiable information such as IP addresses, geographic location, and other personally identifiable information (PII).
Performance efficiency	NFR2	The anonymizer should provide acceptable performance levels, including minimal latency and high throughput, to ensure a smooth user experience. Performance requirements may vary depending on factors such as the number of users, the volume of traffic, and the complexity of anonymization algorithms.
Scalability	NFR3	The anonymizer should be able to scale horizontally or vertically to accommodate increasing user demand and traffic volume without compromising performance or availability. This may involve deploying additional resources dynamically or optimizing resource utilization
Availability	NFR4	should be highly available, with minimal downtime or service interruptions, to ensure continuous access for users
Reliability	NFR5	The anonymizer should be reliable and dependable, consistently delivering accurate anonymization results and maintaining data integrity. This includes error handling mechanisms, data validation checks, and proactive monitoring to detect and address potential issues.
Security	NFR6	employ robust security measures to protect user data from unauthorised access, interception, or tampering
Compliance	NFR7	The anonymizer should comply with relevant legal and regulatory requirements governing data privacy, security, and anonymity. This may include compliance with regulations such as GDPR, CCPA, HIPAA, and industry-specific standards for data protection.
Usability	NFR8	The anonymizer should be user-friendly and easy to use, with intuitive interfaces and clear documentation. Users should be able to configure anonymization settings, monitor system status, and access support resources conveniently.

7.1.6 Component Architecture

The figure below shows the internal architecture of the anonymiser. Its main elements include support for all the functionality described in Section 7.1.7. The anonymizer system is built on a combination of Python and Java technologies, designed to anonymize datasets while connecting to an external Postgres database for data storage during transformations. Key prerequisites include Python with Flask for API functionality, Pandas for data processing, SQLAlchemy for database integration, Presidio for PII identification, Spring Boot for Java-based API services, JPA for database interactions, and ARX Deidentifier for k-Anonymity operations. The architecture centres on a single Docker container hosting a Python API (handling core workflows), a Java Spring Boot API (managing k-Anonymity), and Nginx for routing between APIs and external access. It connects to an external Postgres instance ("Internal Postgres Database"), configured via environment variables (POSTGRES_USER, POSTGRES_PASSWORD, etc.). This database stores dataset metadata and temporary tables, with table names derived from user UUIDs (e.g., _19e923121fba45c692a28ab8bc7882ea). Functionality includes:

- Obfuscation: Applied via mask classes (extending BaseMask), which define transformations per column using properties like name, data_type, and config.
- Sensitivity Reports: Generated by Presidio, classifying columns by PII type (e.g., PHONE_NUMBER) and sensitivity level (e.g., SENSITIVE).
- Synthetic Data: Created using the sdv library to replicate dataset structure.
  k-Anonymity: Implemented via ARX, using quasi-identifiers and generalization hierarchies to anonymize data while assessing re-identification risks through ARX's prosecutor model. The system processes datasets by staging them in Postgres, applying configured masks or k-Anonymity, and outputting anonymized results.

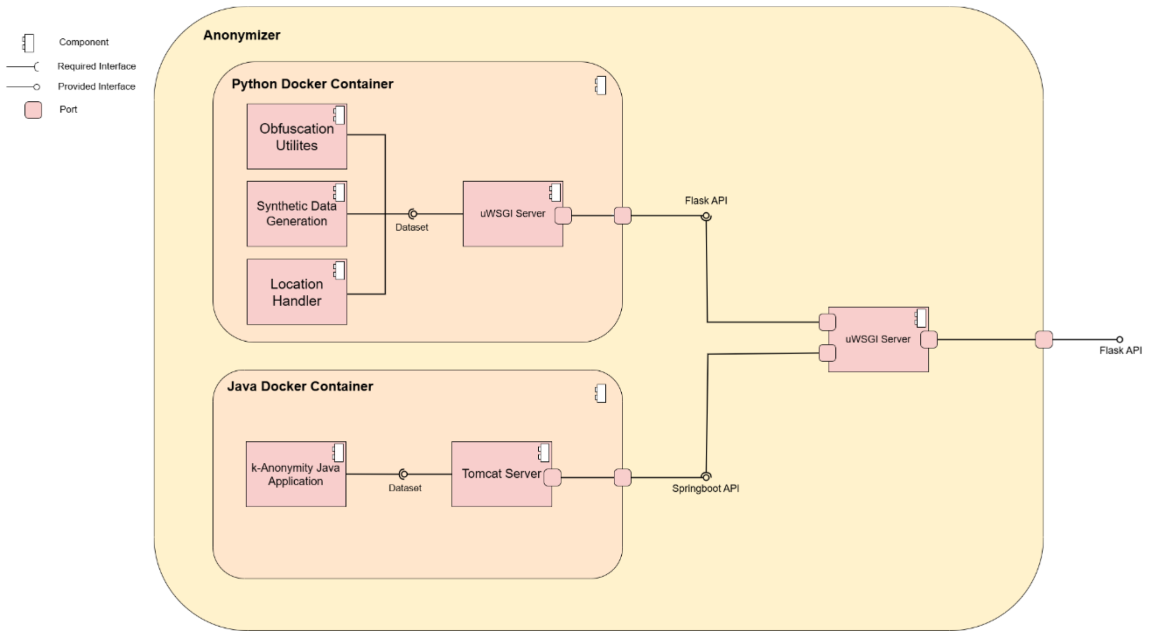


Figure 49: Component's Internal Architecture

7.1.7 Technology Background

The anonymiser consists of a few modularized components. The k-Anonymity provided by the anonymiser will be supported by a dockerized java application using a Spring Boot API. The rest of the anonymisation functionality (deletion, data masking, location anonymisation, differential privacy and pseudonymization) will be supported by a dockerized python application with a Flask API as an interface. These two components will use APIs to communicate internally while a third outward facing Flask API with a uWSGI server provides accessibility to all anonymisation functions from a single API.

7.1.7.1 Anonymiser Technical Details

K-Anonymity

The main functionality of the anonymiser is handled by the ARX data anonymisation java library. The anonymiser uses a k-Anonymity algorithm from the ARX library to remove identifying attributes from a dataset based on parameters provided by the user. This is supported by Jackson to convert the dataset and dataset structure to a format that is readable by ARX.

The API functionality is supported by Spring Boot. This is responsible for deploying the application on port 8080 of the localhost and exposing the API methods for external use.

Deletion

Based on user selection the anonymiser will delete columns and/or rows containing personally identifiable information and/or sensitive data within a dataset.

Data Masking

The Pandas Python Library provides a basic data masking capability. It can be used to obfuscate data by targeting either whole columns or using conditional statements to isolate target values. To increase the variety of masking capabilities we use msticpy to provide hashing functions that we can apply to data while preserving syntax. For example, we may want to provide a hash of an email while preserving its syntax. Msticpy provides syntax preserving hashing for the following formats: string with delimiters; both IPv4 and IPv6 addresses; Several string ID formats such as SID and GUID; Account names while ignoring system names such as root and NT AUTHORITY/SYSTEM; and more. These functions are used both on a single data item or entire DataFrames. These functions are only intended to mask data. No real attempt is made to preserve the syntax and meaning of the output.

Pseudonymisation

This will be supported through the use of the open-source python package called Faker. It is a package that generates fake data by selecting random entries in a database of values based on the category. Faker has a variety of categories such as names, addresses, phone numbers, dates/time, emails, etc.

By creating an instance of the faker generator and selecting a category a pseudonym correlating to that category will be generated and returned for use in the dataset. Below is an example of its use (code in black, results in blue).

```
from faker import Faker
fake = Faker()

fake.name()
'Lucy Cechtelar'

fake.address()
'426 Jordy Lodge, Cartwrightshire, SC 88120-6700'

fake.text()
'Sint velit eveniet. Rerum atque repellat voluptatem quia rerum. Numquam
excepturi beatae sint laudantium consequatur. Magni occaecati itaque sint
et sit tempore. Nesciunt amet quidem. Iusto deleniti cum autem ad quia
aperiam. A consectetur quos aliquam. In iste aliquid et aut similique
suscipit. Consequatur qui quaerat iste minus hic expedita. Consequuntur
error magni et laboriosam. Aut aspernatur voluptatem sit aliquam. Dolores
voluptatum est. Aut molestias et maxime. Fugit autem facilis quos vero.
Eius quibusdam possimus est. Ea quaerat et quisquam. Deleniti sunt quam.
Adipisci consequatur id in occaecati. Et sint et. Ut ducimus quod nemo ab
voluptatum.'
```

Faker Optimisations

The Faker constructor takes a performance-related argument called *use_weighting*. It specifies whether to attempt to have the frequency of values match real-world frequencies (e.g. the English name Gary would be much more frequent than the name Lorimer). If *use_weighting* is *False*, then all items have an equal chance of being selected, and the selection process is much faster. The default is *True*.

7.1.7.2 Anonymiser Location API Technical Details

The location handler is a python-based component responsible for the anonymisation of any columns containing latitude and longitude data. It uses a combination of pandas, numpy and Conditional Tabular Generative Adversarial Networks (CTGAN) from the sdv library to deliver location anonymisation. To provide an internal API to pass data between the Anonymiser it uses Flask to implement an API.

CTGAN is a Generative Adversarial Networks (GAN) based model used to model tabular data distribution and sample rows from the distribution. Its primary focus is generating synthetic data that maintains the trends of the original data whilst removing identifiable features of the original.

GAN algorithms are algorithms typically used for generating synthetic data particularly in video, voice, and image generation. The algorithm works by initialising two separate networks, one called the generator and one called the discriminator. The generator is fed a random input and generates synthetic data based on this input. The discriminator is fed real data and determines whether the synthetic data generated by the generator is fake or real. Based on the judgment given by the discriminator the generator will adjust its output accordingly until the discriminator determines that the generator is able to produce accurate synthetic data.

CTGAN is a GAN-based model focusing on using the same technology for tabular data. The location handler uses a CTGAN algorithm configured at 450 epochs (or training cycles). The location columns are fed to the algorithm as training data then the algorithm generates an equal amount of synthetic data to replace the original location data. This allows it to maintain the same trends whilst randomizing the location to conceal the true identity of the data subject.

7.1.7.3 Synthetic Data Generator

Conditional Tabular GANs (CTGANs) are a specialized class of Generative Adversarial Networks (GANs) designed to generate high-fidelity synthetic tabular data, addressing the unique challenges posed by mixed data types (continuous and categorical) and imbalanced distributions commonly found in real-world datasets.

1. Core Architecture

a. Generator and Discriminator

- Generator: Receives random noise and conditional information (e.g., specific values for categorical columns) as input, and outputs synthetic data samples that mimic the real data distribution.
- Discriminator: Receives both real and synthetic samples, along with their conditional information, and learns to distinguish between authentic and generated data.

b. Conditional Generation

CTGAN conditions both the generator and discriminator on discrete column values, enabling the model to learn complex, multimodal distributions and generate samples for specific subgroups of the data.

2. Data Preparation

- Continuous features should be represented as floats.
- Categorical/discrete features as integers or strings.
- Date features (if present) should be formatted appropriately (e.g., YYYY-MM-DD).
- No missing values: Impute or drop missing data before training.
- Identify discrete columns: Provide a list of categorical features to the model

3. Model Implementation

a. Installation

pip install ctgan

b. Data Loading and Preprocessing

```
import pandas as pd
```

```
from ctgan import CTGAN
# Load your data
data = pd.read_csv('your_data.csv')
# Identify discrete (categorical) columns
discrete_columns = ['col1', 'col2', 'col3']  # Replace with your categorical
columns
```

c. Model Initialization and Training

```
ctgan = CTGAN(epochs=300, batch_size=500, verbose=True)  # Adjust epochs and
batch size as needed
ctgan.fit(data, discrete_columns)
epochs: Number of training iterations; more epochs can improve fidelity but
increase training time.
batch_size: Number of samples per training batch.
```

d. Generating Synthetic Data

```
synthetic_data = ctgan.sample(1000)  # Generate 1000 synthetic samples
```

e. Saving and Loading the Model

```
ctgan.save('ctgan_model.pkl')
# To reload:
from ctgan import CTGAN
ctgan = CTGAN.load('ctgan_model.pkl')
```

4. Underlying Training Process

Adversarial Training: Generator and discriminator are trained in opposition, with the generator aiming to fool the discriminator and the discriminator striving to correctly identify real vs. synthetic samples.

Loss Functions:

- Generator Loss: Encourages the generator to create realistic samples that can deceive the discriminator.
- Discriminator Loss: Trains the discriminator to distinguish between real and fake data.
- Conditional Sampling: During training, CTGAN randomly selects a discrete column and a value to condition on, focusing the generator on learning to produce samples for that subgroup, which helps with imbalanced or rare categories.

5. Advanced Features and Customization

- Handling Date Features: Some CTGAN implementations support cyclical transformations for date/time features.
- Evaluation: Compare statistical properties (e.g., distributions, correlations) between real and synthetic data to assess fidelity.
- Integration: CTGAN can be used as part of broader synthetic data pipelines, including privacy-preserving data sharing, model development, and data augmentation

6. Sample End-to-End Workflow

```
from ctgan import CTGAN
import pandas as pd
# Load and preprocess data
data = pd.read_csv('your_data.csv')
```

```
discrete_columns = ['category1', 'category2']
# Initialize and train CTGAN
ctgan = CTGAN(epochs=100)
ctgan.fit(data, discrete_columns)
# Generate synthetic data
synthetic_data = ctgan.sample(1000)
synthetic_data.to_csv('synthetic_data.csv', index=False)
```

7.1.7.4 Risk of Reidentification

This is being implemented using the open-source libraries provided in ARX.

ARX's risk analysis perspective provides a systematic framework for quantifying and mitigating re-identification risks in anonymised datasets. This functionality is critical for ensuring compliance with privacy regulations like HIPAA and GDPR, as well as for balancing data utility with privacy preservation. Below is a detailed technical breakdown of its methodology, grounded in the tool's documentation and peer-reviewed research.

1. Attacker Models and Risk Scenarios

ARX evaluates re-identification risks through three distinct attacker models, each reflecting different levels of adversarial knowledge and objectives:

a. Prosecutor Scenario

In this model, the attacker knows that a specific individual exists in the dataset and aims to re-identify their record. ARX calculates the highest re-identification risk for any record in the dataset by identifying quasi-identifiers (e.g., age, ZIP code) that uniquely or nearly uniquely distinguish individuals. For example, if a combination of "Age=35, ZIP=90210, Gender=M" appears only once, the risk for that record is 100% under this scenario.

b. Journalist Scenario

Here, the attacker lacks prior knowledge of whether the target individual is in the dataset. Risks are calculated as the probability that a record both matches the attacker's background knowledge and belongs to the target individual. ARX estimates this by factoring in the population prevalence of quasi-identifier combinations. For instance, if a quasi-identifier combination occurs in 0.1% of the population, the risk for a matching record is 0.1%

c. Marketer Scenario

This model assumes the attacker aims to re-identify as many individuals as possible, rather than targeting specific records. ARX computes the average success rate of such attacks by analysing the proportion of records vulnerable to re-identification across the dataset. This metric is particularly useful for assessing risks in marketing or research contexts where bulk data breaches are a concern.

2. Population Uniqueness Estimation

A cornerstone of ARX's risk analysis is its ability to estimate population uniqueness-the likelihood that a record unique in the sample is also unique in the broader population. This is achieved through three statistical super-population models:

- Pitman Model (Hoshino): Suitable for datasets with sampling fractions ≤10%, this method assumes a Pitman-Yor process to estimate the distribution of rare quasi-identifier combinations in the population.
- Zayatz Model: A non-parametric approach that extrapolates population uniqueness directly from sample uniqueness, ideal for datasets with homogeneous attribute distributions.
- Chen & McNulty (SNB): Uses a smoothed bootstrap to account for sampling variability, recommended for clinical or highly skewed datasets.

For example, if a dataset contains 10,000 records and 500 are sample-unique, ARX might estimate (using the Pitman model) that 200 of these are also population-unique, translating to a 4% population uniqueness rate

3. Risk Metrics and Thresholds

ARX computes several key metrics to guide anonymisation decisions:

a. Record-Level Risks

- Highest Risk: Maximum re-identification probability for any individual record (e.g., 25% under the journalist model).
- Lowest Risk: Minimum probability across records (often 0% for well-anonymised data).
- Average Risk: Mean re-identification probability, weighted by record frequency.

b. Threshold Compliance

Users can define acceptable risk thresholds (e.g., “No record may exceed 5% re-identification risk under the prosecutor model”). ARX highlights violations and recommends additional generalization or suppression to meet these criteria.

c. Unique Record Analysis

- Sample Uniques: Records unique within the dataset.
- Estimated Population Uniques: Sample uniques likely unique in the population, calculated via the selected statistical model.

4. Quasi-Identifier Analysis

ARX’s “Finding Quasi-identifiers” view identifies attribute combinations contributing most to re-identification risks. For instance, a dataset might reveal that {ZIP code, Date of Birth, Gender} collectively expose 80% of records to >10% risk under the prosecutor model. The tool ranks these combinations by:

- Separability: How effectively the combination distinguishes records (e.g., entropy-based metrics).
- Distinctness: The number of unique values within the combination.

5. Integration with Anonymisation Workflow

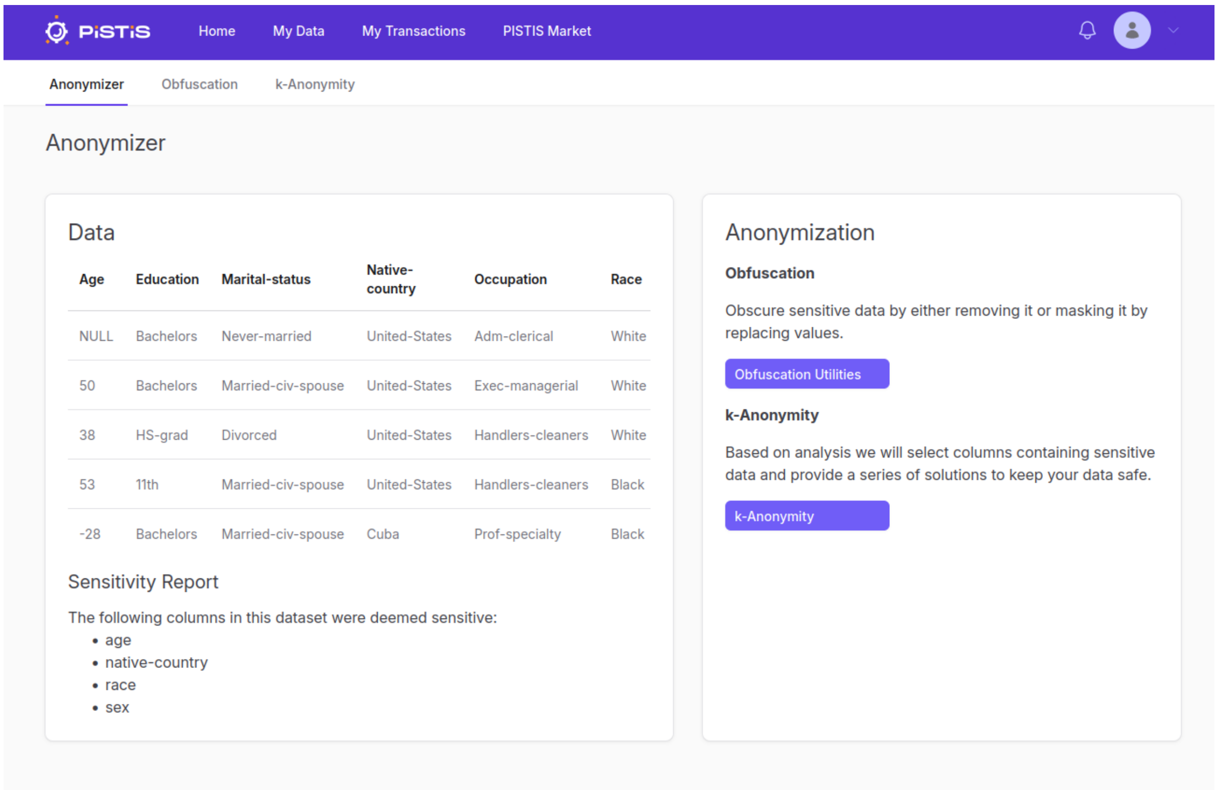
Risk analysis is tightly coupled with ARX’s transformation engine:

- Pre-Anonymisation Baseline: Initial risks are assessed to identify high-risk quasi-identifiers (e.g., “Occupation” may increase average risk from 2% to 8%).

- Post-Anonymisation Validation: After applying k-anonymity or t-closeness, ARX verifies that risks fall below thresholds (e.g., reducing prosecutor risk from 15% to 3%).
- Iterative Refinement: If risks remain high, users can incrementally generalize attributes (e.g., collapsing “Age=35” to “Age=30-40”) and re-evaluate.

7.1.8 Graphical User Interface

The figure below shows the first screen for the anonymiser where the user sees a preview of the data and gets an indication of what the sensitive attributes in the dataset are. They can then from this screen select either data obfuscation or K-Anonymity as anonymisation methods to apply to that dataset.



Anonymizer

Data

Age	Education	Marital-status	Native-country	Occupation	Race
NULL	Bachelors	Never-married	United-States	Adm-clerical	White
50	Bachelors	Married-civ-spouse	United-States	Exec-managerial	White
38	HS-grad	Divorced	United-States	Handlers-cleaners	White
53	11th	Married-civ-spouse	United-States	Handlers-cleaners	Black
-28	Bachelors	Married-civ-spouse	Cuba	Prof-specialty	Black

Sensitivity Report

The following columns in this dataset were deemed sensitive:
- age
- native-country
- race
- sex

Anonymization

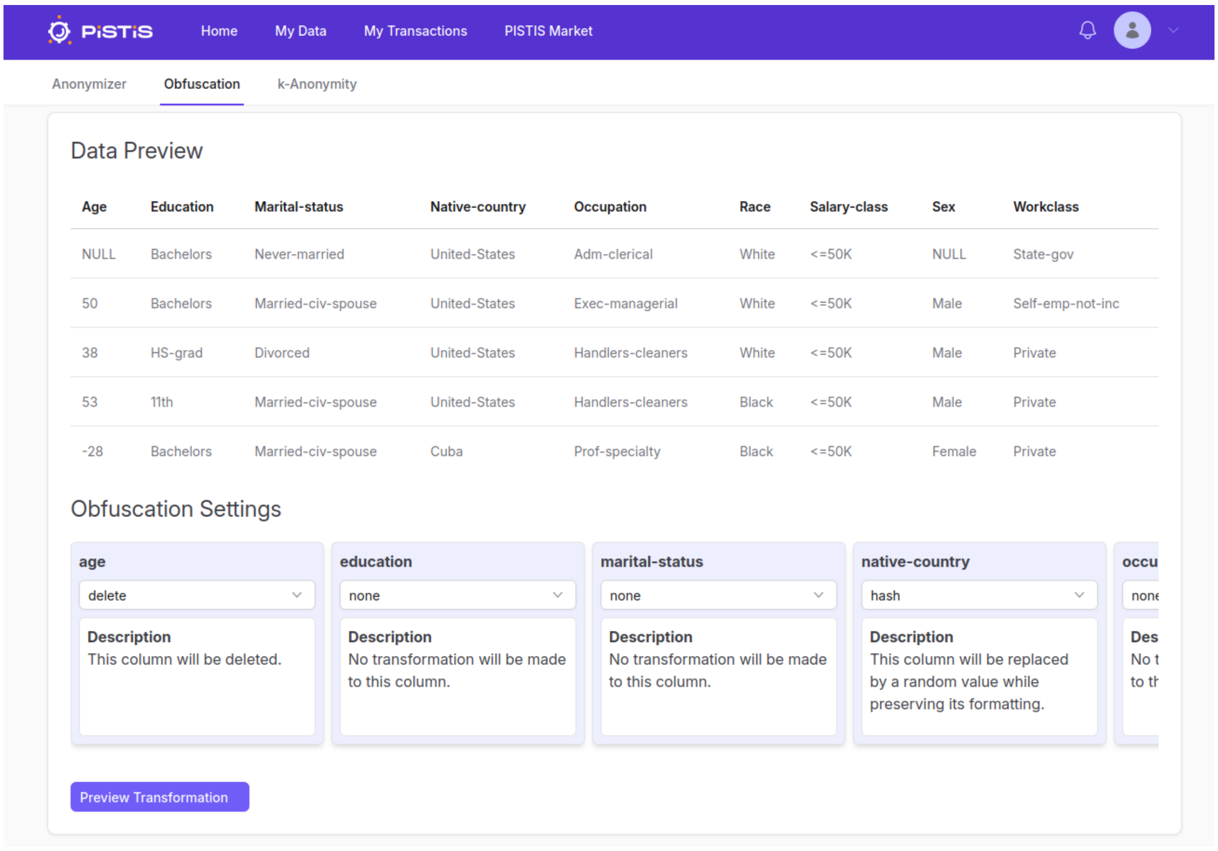
Obfuscation

Obscure sensitive data by either removing it or masking it by replacing values.

k-Anonymity

Based on analysis we will select columns containing sensitive data and provide a series of solutions to keep your data safe.

Figure 50: Initial Anonymiser Screen



Data Preview

Age	Education	Marital-status	Native-country	Occupation	Race	Salary-class	Sex	Workclass
NULL	Bachelors	Never-married	United-States	Adm-clerical	White	<=50K	NULL	State-gov
50	Bachelors	Married-civ-spouse	United-States	Exec-managerial	White	<=50K	Male	Self-emp-not-inc
38	HS-grad	Divorced	United-States	Handlers-cleaners	White	<=50K	Male	Private
53	11th	Married-civ-spouse	United-States	Handlers-cleaners	Black	<=50K	Male	Private
-28	Bachelors	Married-civ-spouse	Cuba	Prof-specialty	Black	<=50K	Female	Private

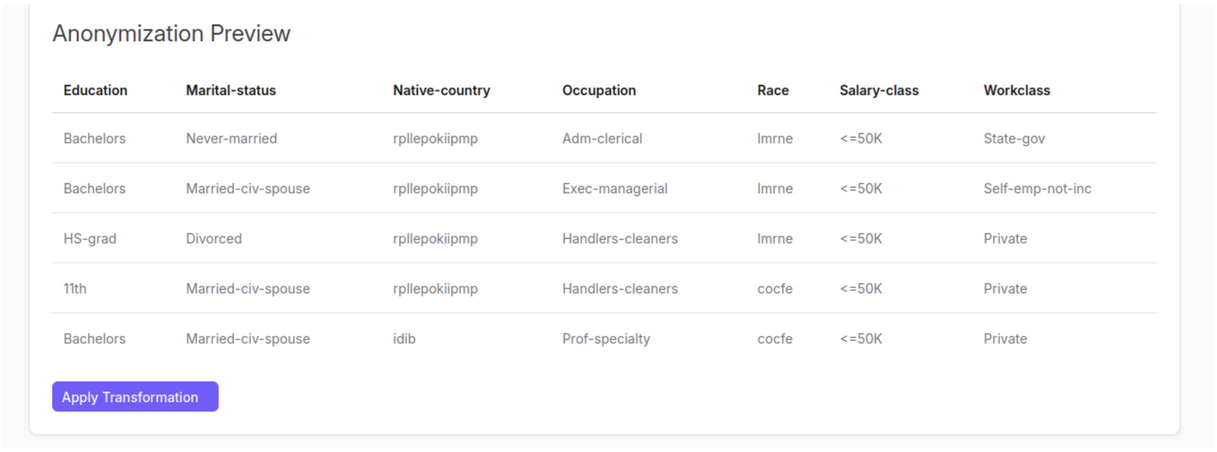
Obfuscation Settings

age	education	marital-status	native-country	occu
delete	none	none	hash	none
Description This column will be deleted.	Description No transformation will be made to this column.	Description No transformation will be made to this column.	Description This column will be replaced by a random value while preserving its formatting.	Des No t to th

Preview Transformation

Figure 51: Anonymiser Obfuscation Utilities

The figure above shows the data preview and the suggested obfuscation settings based on what the anonymisation detects to be sensitive attributes. The user can change the obfuscation settings if they wish to diverge from the suggested settings.



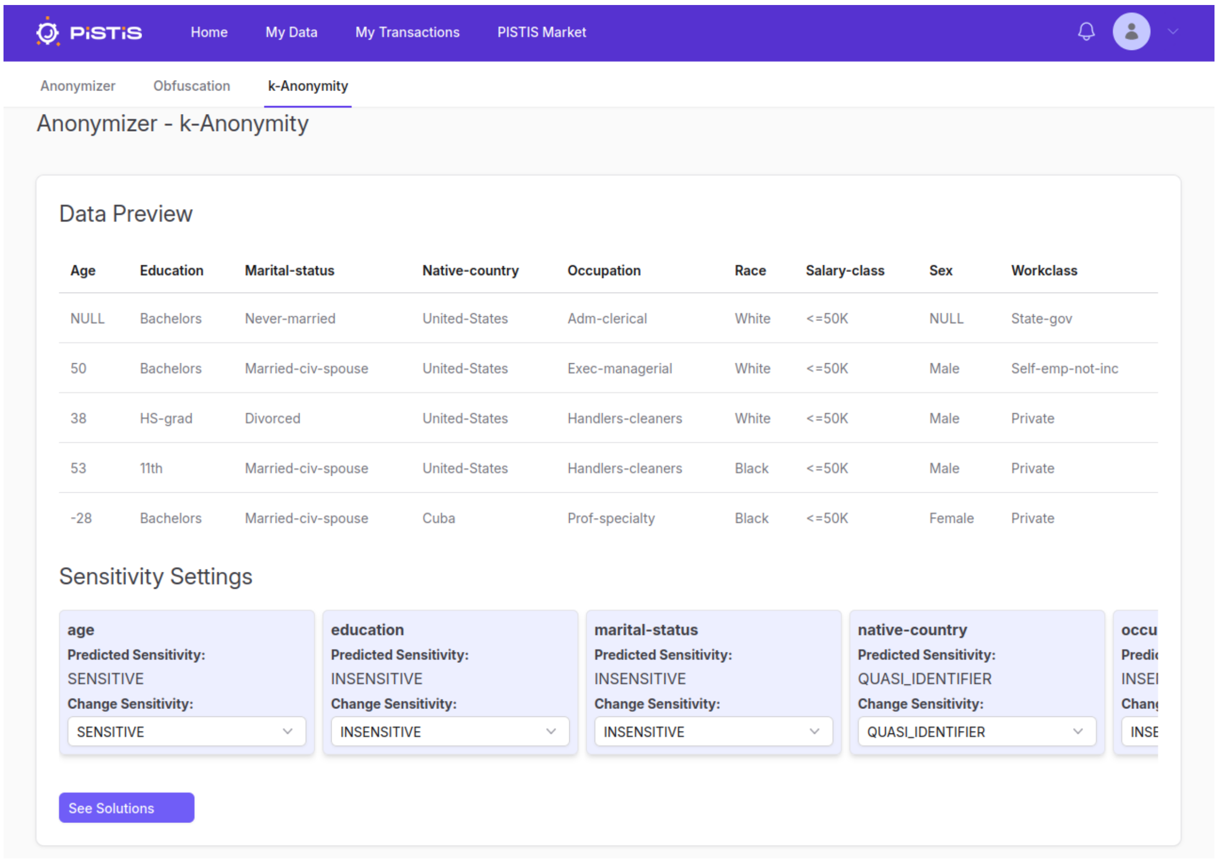
Anonymization Preview

Education	Marital-status	Native-country	Occupation	Race	Salary-class	Workclass
Bachelors	Never-married	rpllepokiipmp	Adm-clerical	lmrne	<=50K	State-gov
Bachelors	Married-civ-spouse	rpllepokiipmp	Exec-managerial	lmrne	<=50K	Self-emp-not-inc
HS-grad	Divorced	rpllepokiipmp	Handlers-cleaners	lmrne	<=50K	Private
11th	Married-civ-spouse	rpllepokiipmp	Handlers-cleaners	cocfe	<=50K	Private
Bachelors	Married-civ-spouse	idib	Prof-specialty	cocfe	<=50K	Private

Apply Transformation

Figure 52: Obfuscation Anonymiser Preview

In the figure above the user sees a preview of the anonymised dataset once they have selected the obfuscation settings and decide to review the resulting transformation to the dataset.



Anonymizer - k-Anonymity

Data Preview

Age	Education	Marital-status	Native-country	Occupation	Race	Salary-class	Sex	Workclass
NULL	Bachelors	Never-married	United-States	Adm-clerical	White	<=50K	NULL	State-gov
50	Bachelors	Married-civ-spouse	United-States	Exec-managerial	White	<=50K	Male	Self-emp-not-inc
38	HS-grad	Divorced	United-States	Handlers-cleaners	White	<=50K	Male	Private
53	11th	Married-civ-spouse	United-States	Handlers-cleaners	Black	<=50K	Male	Private
-28	Bachelors	Married-civ-spouse	Cuba	Prof-specialty	Black	<=50K	Female	Private

Sensitivity Settings

age
Predicted Sensitivity: SENSITIVE
Change Sensitivity: SENSITIVE

education
Predicted Sensitivity: INSENSITIVE
Change Sensitivity: INSENSITIVE

marital-status
Predicted Sensitivity: INSENSITIVE
Change Sensitivity: INSENSITIVE

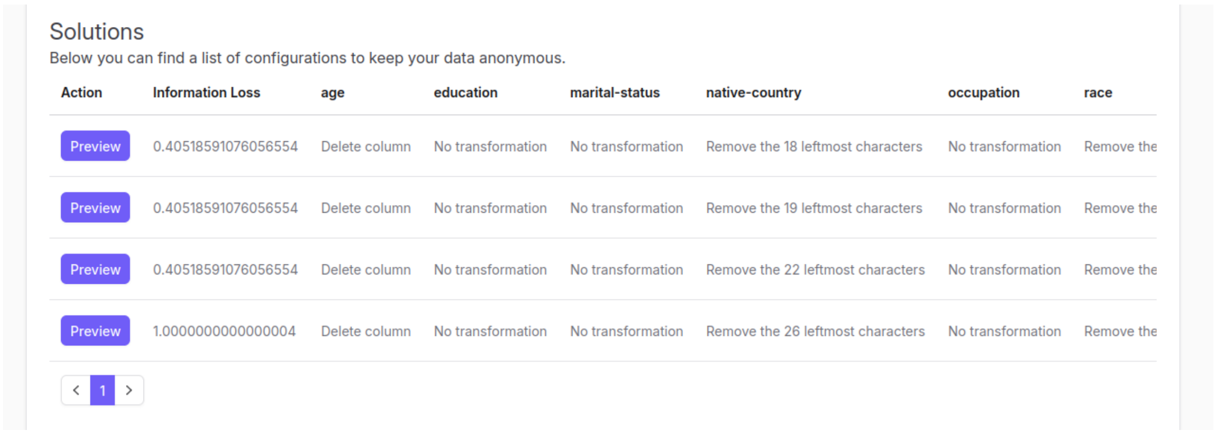
native-country
Predicted Sensitivity: QUASI_IDENTIFIER
Change Sensitivity: QUASI_IDENTIFIER

occu
Predic
INSEI
Chan
INSE

See Solutions

Figure 53: Anonymiser K-Anonymity

The figure above shows the screen for K-Anonymity and the user is shown the sensitivity settings again for each attribute. They can then select to see possible anonymisation solutions



Solutions

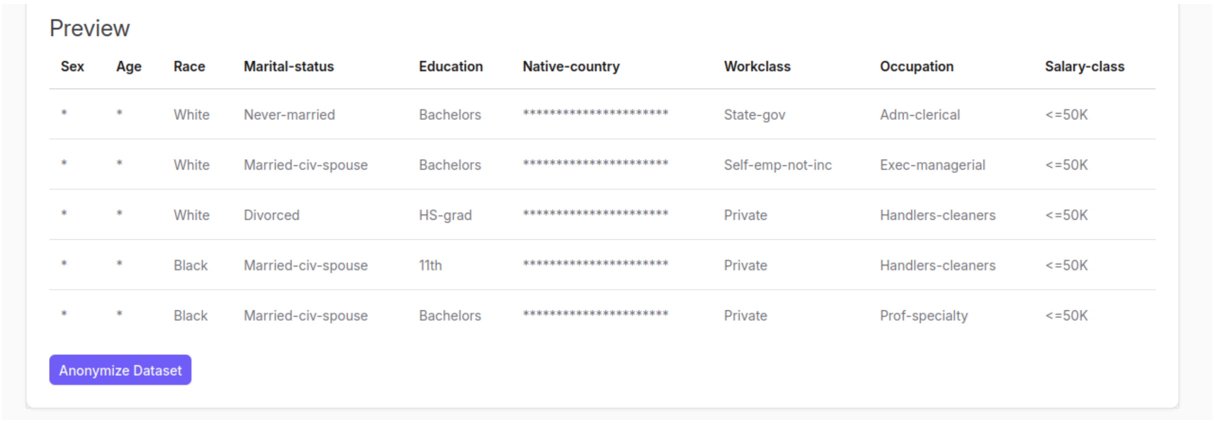
Below you can find a list of configurations to keep your data anonymous.

Action	Information Loss	age	education	marital-status	native-country	occupation	race
Preview	0.40518591076056554	Delete column	No transformation	No transformation	Remove the 18 leftmost characters	No transformation	Remove the
Preview	0.40518591076056554	Delete column	No transformation	No transformation	Remove the 19 leftmost characters	No transformation	Remove the
Preview	0.40518591076056554	Delete column	No transformation	No transformation	Remove the 22 leftmost characters	No transformation	Remove the
Preview	1.0000000000000004	Delete column	No transformation	No transformation	Remove the 26 leftmost characters	No transformation	Remove the

< 1 >

Figure 54: Anonymiser K-Anonymity Solutions

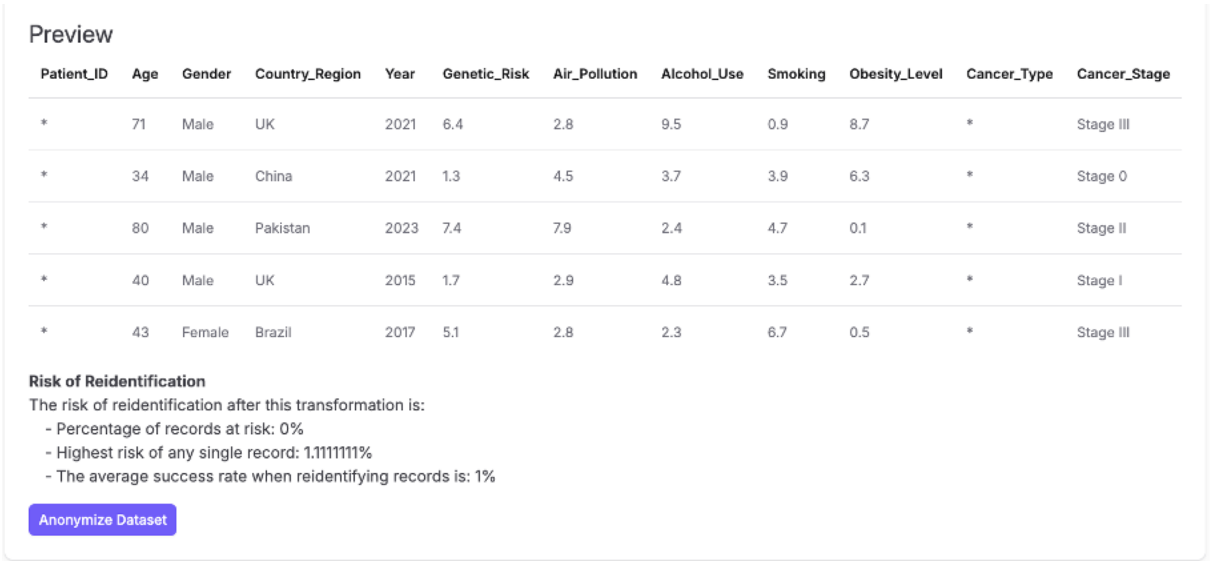
In the figure above the possible anonymisation solutions are shown and in the figure below the user sees a preview of a dataset once a specific solution from the ones shown above is selected.


Preview

Sex	Age	Race	Marital-status	Education	Native-country	Workclass	Occupation	Salary-class
*	*	White	Never-married	Bachelors	**********************	State-gov	Adm-clerical	<=50K
*	*	White	Married-civ-spouse	Bachelors	**********************	Self-emp-not-inc	Exec-managerial	<=50K
*	*	White	Divorced	HS-grad	**********************	Private	Handlers-cleaners	<=50K
*	*	Black	Married-civ-spouse	11th	**********************	Private	Handlers-cleaners	<=50K
*	*	Black	Married-civ-spouse	Bachelors	**********************	Private	Prof-specialty	<=50K

Anonymize Dataset

Figure 55: Preview of Dataset after K-Anonymity Solution is selected


Preview

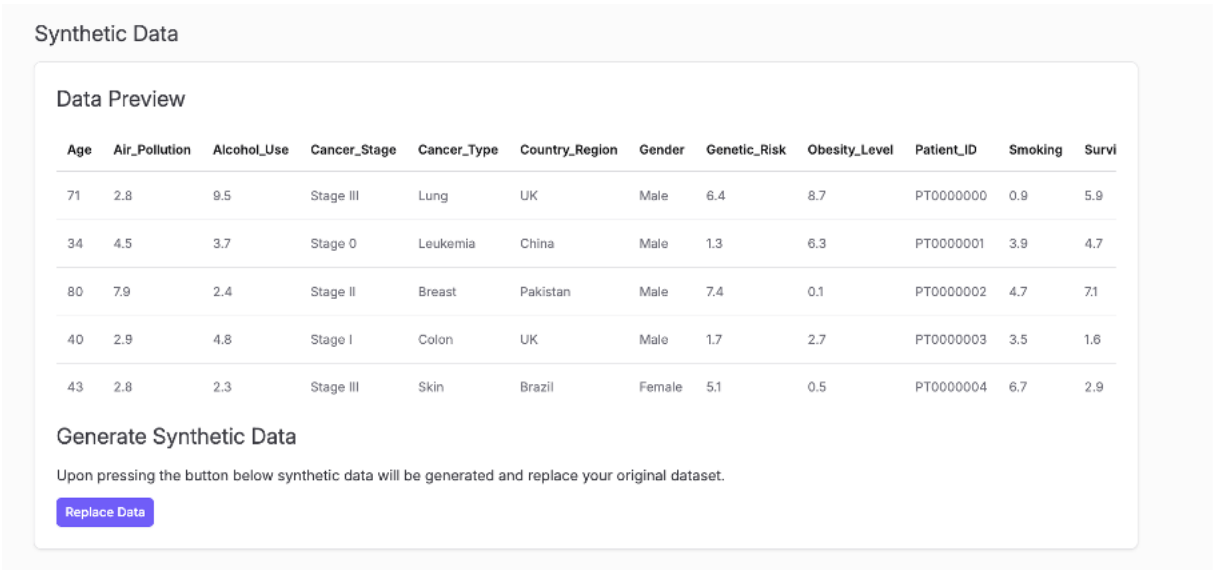
Patient_ID	Age	Gender	Country_Region	Year	Genetic_Risk	Air_Pollution	Alcohol_Use	Smoking	Obesity_Level	Cancer_Type	Cancer_Stage
*	71	Male	UK	2021	6.4	2.8	9.5	0.9	8.7	*	Stage III
*	34	Male	China	2021	1.3	4.5	3.7	3.9	6.3	*	Stage 0
*	80	Male	Pakistan	2023	7.4	7.9	2.4	4.7	0.1	*	Stage II
*	40	Male	UK	2015	1.7	2.9	4.8	3.5	2.7	*	Stage I
*	43	Female	Brazil	2017	5.1	2.8	2.3	6.7	0.5	*	Stage III

Risk of Reidentification
The risk of reidentification after this transformation is:
- Percentage of records at risk: 0%
- Highest risk of any single record: 1.1111111%
- The average success rate when reidentifying records is: 1%

Anonymize Dataset

Figure 56: Preview of a dataset after K-Anonymity has been selected with metrics for Risk of Reidentification

The figure above also shows the risk of re-identification metrics – these are shown when the user previews a solution informing the user of the privacy risk that still remains if that solution is the one that is chosen.


Synthetic Data

Data Preview

Age	Air_Pollution	Alcohol_Use	Cancer_Stage	Cancer_Type	Country_Region	Gender	Genetic_Risk	Obesity_Level	Patient_ID	Smoking	Survi
71	2.8	9.5	Stage III	Lung	UK	Male	6.4	8.7	PT0000000	0.9	5.9
34	4.5	3.7	Stage 0	Leukemia	China	Male	1.3	6.3	PT0000001	3.9	4.7
80	7.9	2.4	Stage II	Breast	Pakistan	Male	7.4	0.1	PT0000002	4.7	7.1
40	2.9	4.8	Stage I	Colon	UK	Male	1.7	2.7	PT0000003	3.5	1.6
43	2.8	2.3	Stage III	Skin	Brazil	Female	5.1	0.5	PT0000004	6.7	2.9

Generate Synthetic Data

Upon pressing the button below synthetic data will be generated and replace your original dataset.

Replace Data

Figure 48: Synthetic Data Generator

The figure above shows the data preview for the synthetic data generation option. Synthetic data can then be generated to replace the original dataset if the user wants to further reduce any privacy risk.

7.1.9 Developer Documentation

Documentation for developers is available here:

https://docs.pistis-market.eu/developers/anonymiser/anonymiser

7.1.10 Source Code

Anonymizer is a proprietary tool owned and developed by Assentian. The source code is managed in a private project code repository[11]. The deployment branch contains the code currently deployed on the PISTIS Kubernetes cluster[12].

7.2 Lineage Tracker

7.2.1 Component Description

The Lineage Tracker records every create, update, and delete operation performed on a dataset, documenting the actor, timestamp, and nature of each change. Each operation is then translated into RDF according to the W3C PROV-Ontology and persisted in a triple store, incrementally constructing a comprehensive lineage graph of the dataset's evolution. Upon each update, the system computes the diff between successive versions and stores a textual summary of the modifications made. Additionally, a cryptographic hash of this updated lineage snapshot is computed and anchored on a blockchain to ensure an immutable, verifiable audit trail. This comprehensive lineage history grants users full visibility into a dataset's evolution as well as access to any prior version.

The Lineage Tracker offers API endpoints for logging create, update, and delete operations on a dataset; retrieving structured lineage data; and computing diffs between any two dataset versions. Each incoming operation is serialized into RDF and persisted in a triple store, thereby incrementally extending the dataset's lineage graph. Clients may then query the API to obtain a dataset's full history, its lineage graph, the complete history of a particular user, or the diff between two dataset versions.

[11] https://github.com/PISTIS-Platform/Anonymiser
[12] https://github.com/PISTIS-Platform/Anonymiser/tree/deployment

Lastly, the Lineage Tracker provides a user-friendly user interface (UI) with two primary pages. The Lineage History page displays an interactive lineage tree of all dataset versions alongside a detailed history table, listing each operation's timestamp, actor, and operation description. The Dataset Versions page enables users to select any two dataset versions and review their differences in a comparison table, detailing all additions, deletions, and modifications.

7.2.2 Main Improvements in Beta Version

The major Beta Version improvements include the following:

Version diff: When two dataset versions are selected in the UI Compare Versions page, the Lineage Tracker computes the diff between them. Any schema and data differences between the two versions is then displayed in the UI.

Textual update description: When a dataset is updated, the Lineage Tracker detects any schema changes, data changes, data enrichments, and data transformations made to the dataset. Notably, the system detects any of the data transformations supported in the PISTIS Transformations catalogue (datetime harmonization, removing missing registries, replacing values, replacing missing registries with fixed or various statistical methods). It then stores a textual description of these changes in RDF and displays it in the UI.

Cloud Catalogue Lineage Tracker: When a Data Owner places a dataset up for sale, the dataset's lineage information is transferred to the cloud triple store, and can be viewed in the cloud Lineage Tracker UI. The cloud Lineage Information page displays the same information as the factory Lineage Information Page; however, the cloud Dataset Versions page only displays a sample of dataset changes when a user clicks on two dataset versions.

Blockchain anchoring: Whenever a dataset is created, updated, or deleted, a cryptographic hash is computed of the dataset's lineage information, and stored on the Blockchain component. This ensures the lineage information is not tampered with, as its hash can easily be verified on the Blockchain.

Enhanced user interface: The Lineage Tracker UI user interface was improved, particularly the styling with regards to the dataset's family tree, information table, and dataset comparison table.

Dataset anonymization support: The Lineage Tracker was updated to allow for anonymized datasets. When a dataset is anonymized, the visibility of its predecessor datasets is updated so that cloud catalogue users cannot view pre-anonymized dataset differences when comparing dataset versions.

7.2.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	Data Consumer/ Data Provider	Explore the version history of a dataset.	I get an overview of the operations performed on it/I can see how my dataset performs (is being used).	Alpha (Backend) Beta (UI)	Data Consumer can view all versions of a dataset and all CRUD operations performed on it.	Done	PISTIS. OUS. 07
UC_02	Data Provider	Access previous version(s) of a dataset.	I can use the specific version, which suits my needs for further purposes.	Alpha (Backend) Beta (UI)	Data Provider can access all versions of a dataset.	Done	PISTIS. OUS. 08
UC_03	Data Consumer	Verify who is modifying my dataset.	I have visibility over who is modifying my data.	Alpha (Backend) Beta (UI)	Data Provider can view all Data Consumer accessing dataset.	Done	PISTIS. OUS. 08

7.2.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	Document (via API) and on request provide (via API and UI) the information that a new dataset was checked in.	PISTIS.OUS.01	
FR_02	Document (via API) and on request provide (via API and UI) the information that a dataset was modified.	PISTIS.OUS.03	
FR_03	Provide data lineage information to contribute to the quality assessment of a dataset.	PISTIS.OUS.04	
FR_04	Provide data lineage information to the data valuation service (via API).	PISTIS.OUS.07	
FR_05	Provide data lineage information to the analytics engine (via API).	PISTIS.OUS.08	

7.2.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Functional Suitability	NFR1	The Lineage Tracker adheres to all functional requirements.
Performance efficiency	NFR2	The Lineage Tracker stores and retrieves lineage information, maintaining high throughput and low latency in data processing.

Compatibility	NFR3	The Lineage Tracker can be integrated with any PISTIS components that need to access the operation documentation and information retrieval API endpoints.
Usability	NFR4	The Lineage Tracker allows users to view lineage information in an intuitive, user-friendly user interface.
Reliability	NFR5	The Lineage Tracker guarantees all API endpoints are available with minimal downtime. Proper error messages are provided with all failed requests.
Security	NFR6	The Lineage Tracker guarantees only the Factory Data Storage can access Operation Documentation API endpoints and only authorized users can access Information Retrieval API endpoints.
Portability	NFR7	The Lineage Tracker is containerized and can be deployed across different operating systems and environments.

7.2.6 Component Architecture

Per Figure 57 below, the Lineage Tracker consists of the User Interface, API, and Backend.

The User Interface (UI) is used to visualize the lineage information using Vue.js and Pinia for frontend state management. When a user wants to view a dataset's lineage information, they can navigate to a dataset's lineage page, which in turn requests the dataset's lineage data from the API. Similarly, when a user wants to compare two dataset versions, they can click on two datasets in the family tree, which in turn requests dataset diff data from the API. The Vue.js UI then displays this information in the form of a dataset family tree and tables.

The Flask REST API serves as the interface to both document dataset lineage events and to subsequently retrieve this lineage information. The API is divided into two main modules: the Operation Documentation and Information Retrieval modules. Whenever a dataset is modified, the Factory Data Storage calls the Operation Documentation module which captures and records dataset changes. The UI can then call the Information Retrieval module to retrieve a dataset's lineage information for visualization purposes.

The Backend consists of a Virtuoso Triplestore which is used to store all lineage data in RDF and an Indexing Service which converts RDF to JSON and vice versa. By leveraging the W3C Prov-DM, the Virtuoso Triplestore can model complex relationships between users, datasets, and operations—namely create, update, and delete. The Indexing Service then serves as middleware between the REST API and Virtuoso Triplestore, using the python-prov library to convert json data from the Factory Data Storage to RDF that can be stored in the triplestore, and to convert RDF in the triplestore to JSON that can be used by the UI.

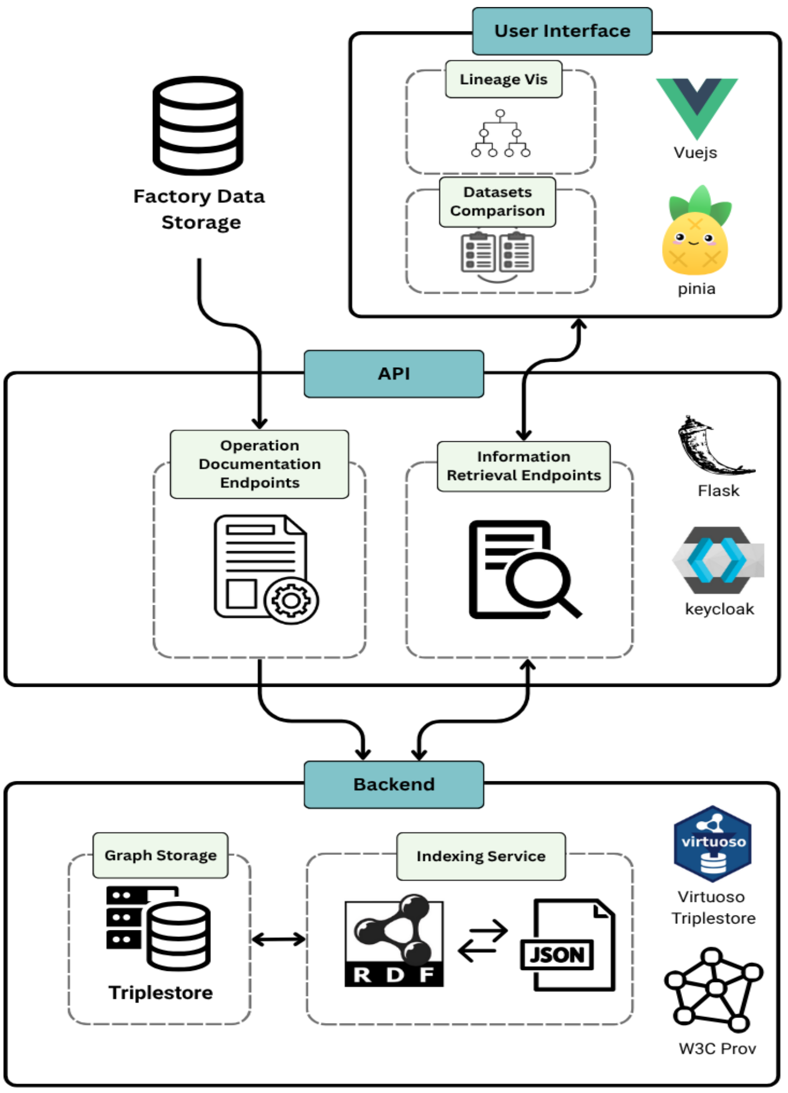


Figure 57: Lineage Tracker Architecture Diagram

7.2.7 Technology Background

The Lineage Tracker consists of a REST API for documenting operations and retrieving lineage information, a database for storing this lineage information, and a frontend UI for visualizing it.

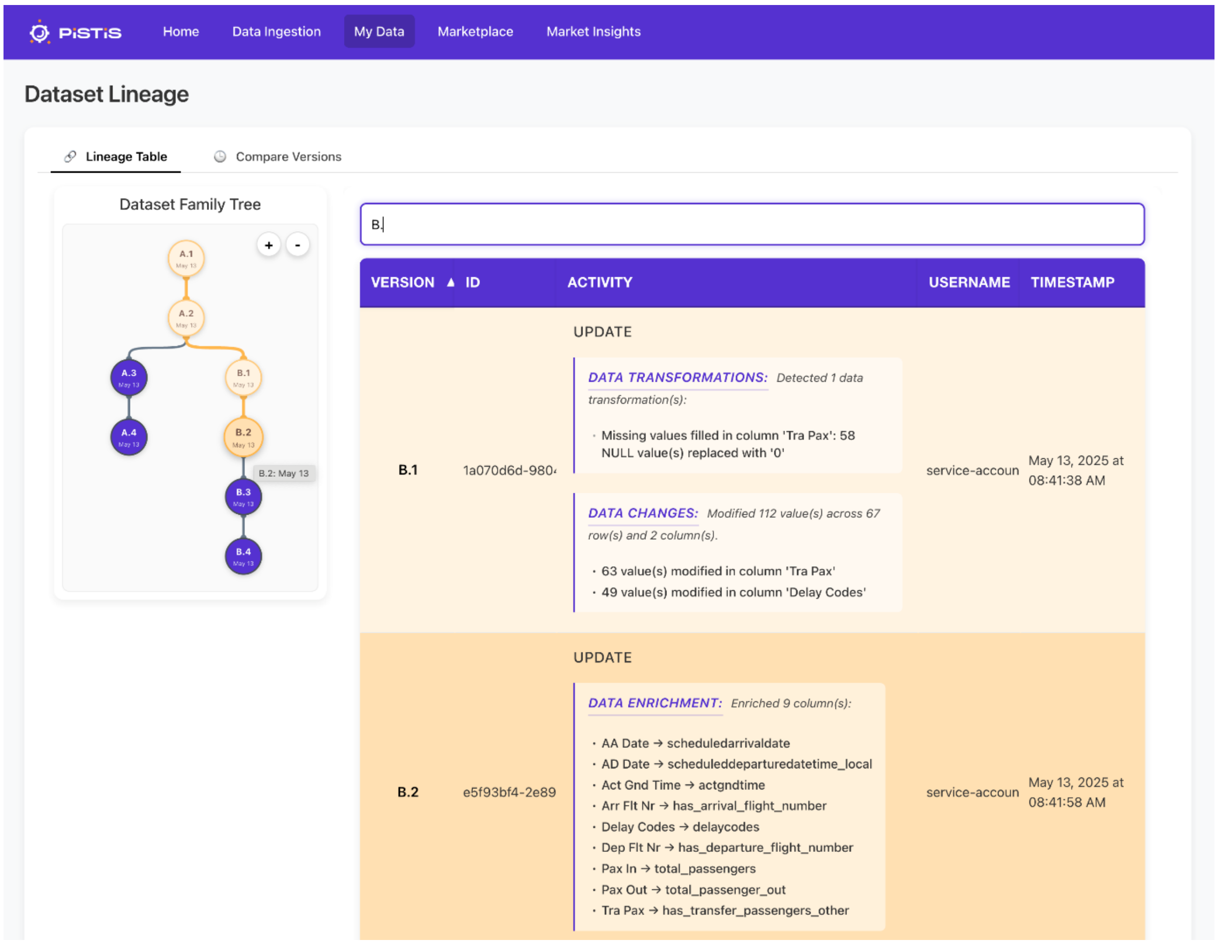
The REST API is built in Python using the Flask framework to realize the micro service architecture and python-prov library for managing lineage data. Flask is a lightweight and flexible web application framework for Python that allows for the quick development of extensible and modular REST APIs. Furthermore, python-prov is a Python library implementation of the W3C PROV-Ontology, allowing for the creation, manipulation, and export of lineage data according to the W3C PROV Data Model.

When a dataset's lineage is constructed, the resulting RDF graph is stored in an Openlink Virtuoso triple store, where it can be updated and queried. An Openlink Virtuoso triple store is a high-performance database designed to store and manage RDF data. Furthermore, it supports SPARQL queries for accessing data, making it suitable for storing and retrieving complex lineage information.

Lastly, the frontend UI is realized using Vue.js, an open-source JavaScript framework used for building user interfaces in single-page applications. In addition, the frontend employs Pinia for state management and Bootstrap for responsive CSS styling.

7.2.8 Graphical User Interface

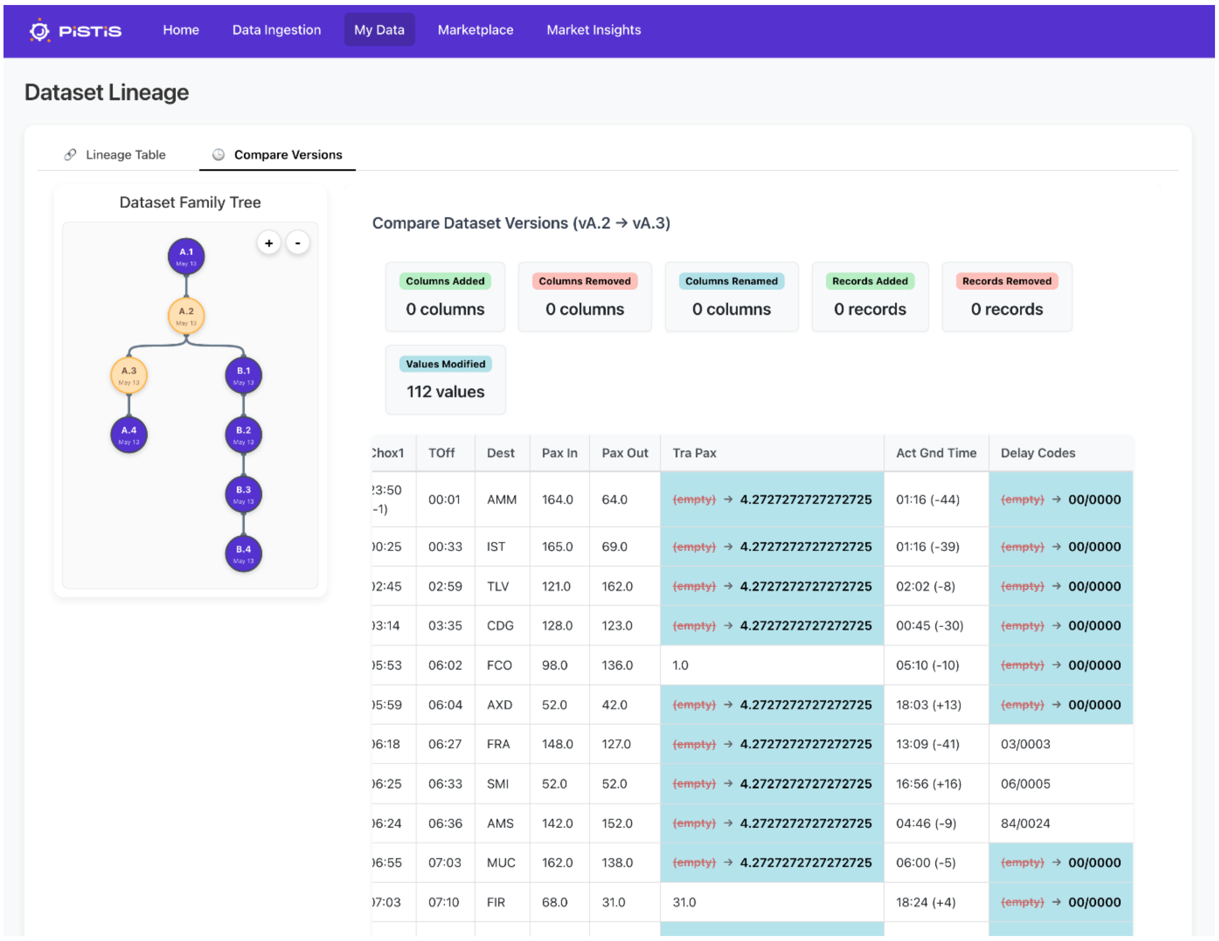
The screenshot below presents a dataset's lineage: on the left, a graph illustrates its family tree; on the right, a table presents more detailed lineage information.



VERSION ▲	ID	ACTIVITY	USERNAME	TIMESTAMP
B.1	1a070d6d-980	UPDATE *DATA TRANSFORMATIONS: Detected 1 data transformation(s):* • Missing values filled in column 'Tra Pax': 58 NULL value(s) replaced with '0' *DATA CHANGES: Modified 112 value(s) across 67 row(s) and 2 column(s).* • 63 value(s) modified in column 'Tra Pax' • 49 value(s) modified in column 'Delay Codes'	service-accoun	May 13, 2025 at 08:41:38 AM
B.2	e5f93bf4-2e89	UPDATE *DATA ENRICHMENT: Enriched 9 column(s):* • AA Date → scheduledarrivaldate • AD Date → scheduleddeparturedatetime_local • Act Gnd Time → actgndtime • Arr Flt Nr → has_arrival_flight_number • Delay Codes → delaycodes • Dep Flt Nr → has_departure_flight_number • Pax In → total_passengers • Pax Out → total_passenger_out • Tra Pax → has_transfer_passengers_other	service-accoun	May 13, 2025 at 08:41:58 AM

Figure 58: View Dataset Lineage

The screenshot below compares two datasets: on the left, a graph illustrates their family tree; on the right, a table highlights their differences.



Compare Dataset Versions (vA.2 → vA.3)

Columns Added	Columns Removed	Columns Renamed	Records Added	Records Removed	Values Modified
0 columns	0 columns	0 columns	0 records	0 records	112 values

Chox1	TOff	Dest	Pax In	Pax Out	Tra Pax	Act Gnd Time	Delay Codes
23:50 -1)	00:01	AMM	164.0	64.0	~~(empty)~~ → 4.2727272727272725	01:16 (-44)	~~(empty)~~ → 00/0000
00:25	00:33	IST	165.0	69.0	~~(empty)~~ → 4.2727272727272725	01:16 (-39)	~~(empty)~~ → 00/0000
02:45	02:59	TLV	121.0	162.0	~~(empty)~~ → 4.2727272727272725	02:02 (-8)	~~(empty)~~ → 00/0000
03:14	03:35	CDG	128.0	123.0	~~(empty)~~ → 4.2727272727272725	00:45 (-30)	~~(empty)~~ → 00/0000
05:53	06:02	FCO	98.0	136.0	1.0	05:10 (-10)	~~(empty)~~ → 00/0000
05:59	06:04	AXD	52.0	42.0	~~(empty)~~ → 4.2727272727272725	18:03 (+13)	~~(empty)~~ → 00/0000
06:18	06:27	FRA	148.0	127.0	~~(empty)~~ → 4.2727272727272725	13:09 (-41)	03/0003
06:25	06:33	SMI	52.0	52.0	~~(empty)~~ → 4.2727272727272725	16:56 (+16)	06/0005
06:24	06:36	AMS	142.0	152.0	~~(empty)~~ → 4.2727272727272725	04:46 (-9)	84/0024
06:55	07:03	MUC	162.0	138.0	~~(empty)~~ → 4.2727272727272725	06:00 (-5)	~~(empty)~~ → 00/0000
07:03	07:10	FIR	68.0	31.0	31.0	18:24 (+4)	~~(empty)~~ → 00/0000

Figure 59: View Dataset Version Changes

7.2.9 Developer Documentation

Documentation and guides are available at https://docs.pistis-market.eu/developers/lineage-tracker/introduction, which provides support for implementation and integration with other components.

7.2.10 Source Code

Lineage Tracker has a frontend module built in Vue.js and a backend module built in Python. The source code of these modules is available in the PISTIS GitHub under this repo: https://github.com/PISTIS-Platform/lineage-tracker . The user interface of lineage tracker can be triggered from a dataset page in the Factory Catalogue UI and the Swagger UI of the backend is available here: https://develop.pistis-market.eu/srv/lineage-tracker

7.3 GDPR checker

7.3.1 Component Description

The GDPR Checker is invoked automatically whenever a dataset is uploaded to the PISTIS platform. It employs ontologies—machine-readable documents in OWL/ODRL format—that define GDPR concepts (for example, “personal data” and “sensitive data”). Each column within the dataset is annotated with a label (for instance, “email” or “medical_record”). Upon submission of a dataset, the GDPR Checker retrieves and loads the relevant ontology files, which specify the applicable GDPR rules.

Subsequently, it conducts a comparison between the dataset’s labels and the definitions contained in these ontologies. For example, if a column labelled “email” is found alongside one labelled “health_diagnosis,” the Checker, by virtue of the ontology relationships, recognizes that linking those two fields may contravene anonymity requirements. In such circumstances, the GDPR Checker generates a warning—such as “Email addresses combined with health information may constitute an anonymity breach”—and provides a clear recommendation, for example, “Apply anonymisation to the ‘email’ column using the Anonymizer” or “Remove ‘health_diagnosis’ if it is not strictly necessary.”

Finally, the component produces a concise report indicating either “No issues found” or “Possible GDPR concerns,” accompanied by an itemized list of warnings and corresponding mitigation suggestions. This report is forwarded to the PISTIS user interface so that users can immediately identify and address any potential risks before the data proceeds downstream. By leveraging ontologies and formalized rules, the GDPR Checker Beta functions as an early warning system that notifies users of potential GDPR issues without issuing a definitive legal compliance certificate.

7.3.2 Main Improvements in Beta Version

Below is a brief overview of the principal enhancements introduced in the Beta version of the GDPR Checker:

1. Ontology-Based Evaluation: Replaces the simple rule-based engine with machine-readable OWL/ODRL ontologies that formally define GDPR concepts (e.g., “personal data,” “sensitive data,” “anonymity”).

2. Dynamic Rule Loading: Ontologies and GDPR rules are loaded at runtime, allowing easier updates and extensibility compared to hardcoded rule sets.

3. Advisory Output Model: Instead of generating a binary “compliant/not-compliant” certificate, the Beta Checker issues targeted warnings and mitigation suggestions (e.g., “apply anonymisation to ‘email’,” “remove ‘health_diagnosis’”).

4. Fine-Grained Privacy Profile Comparison: Directly compares dataset labels against ontology relationships, enabling detection of specific conflicts (such as linking email with health data) rather than relying on broad rule combinations.

5. Early-Warning Integration: Runs automatically when a dataset is uploaded and feeds its report to the PISTIS UI, ensuring that potential GDPR issues are flagged before any downstream processing.

6. Improved Extensibility: Partners and legal teams can supply or update ontology files without modifying core code, supporting evolving legal requirements.

7. Streamlined Workflow: Eliminates the previous step of generating smart contracts for certification, focusing instead on immediate, actionable feedback.

7.3.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
US_01	Data Provider	know whether my data/dataset comply with the GDPR regulation (according to generic rules	make the necessary actions (e.g., anonymisation s)	Alpha	GDPR compliance report	Done	PISTIS. OUS.4 & PISTIS. OUS.7
US_02	Data Consumer	rest assured that the data I acquired are GDPR compliant (according to generic rules)	no legal issues occur	Alpha	GDPR compliance report	Done	PISTIS. OUS.4 & PISTIS. OUS.7
US_03	Data Provider	know whether my data/dataset comply with the GDPR regulation (according to limited GDPR articles)	make the necessary actions (e.g., anonymisation s)	Beta	GDPR report	Done	PISTIS. OUS.4 & PISTIS. OUS.7
US_04	Data Consumer	rest assured that the data I acquired have no GDPR warnings (according to limited GDPR articles)	no legal issues occur	Beta	GDPR report	Done	PISTIS. OUS.4 & PISTIS. OUS.7

7.3.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	GDPR compliance check process for a specified dataset**.**	UC_1, UC_2	This compliance report will be considered in the Data Valuation process.
FR_02	Possible GDPR warnings and recommendations are stored in the Public Ledger as a proof of	UC_1, UC_2	Other PISTIS components can check the dataset's GDPR status.

	compliance.		
FR_03	If dataset is not GDPR compliant potential mitigation measures in the form of suggestions will be provided to the user.	UC_1, UC_2	The user will be notified with potential GDPR warnings in order to take action.

7.3.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Performance efficiency	NFR1	GDPR compliance check should be performed in efficient way.
Usability	NFR2	Authorized entities should easily check GDPR compliance through the Public Ledger stored certificate.
Reliability	NFR3	Compliance results should be reliable and displayed only to authorised entities.
Security	NFR4	Only authorized entities should check GDPR report and it should be stored in a way this could not be altered.

7.3.6 Component Architecture

The GDPR Checker verifies GDPR compliance through a clear process. It first identifies sensitive data and relevant privacy policies in the dataset using two dedicated modules. These results feed into the GDPR Check module, which evaluates if the dataset meets standards of the applied rules. The component then outputs potential GDPR Warnings and recommendations for the user to take action.

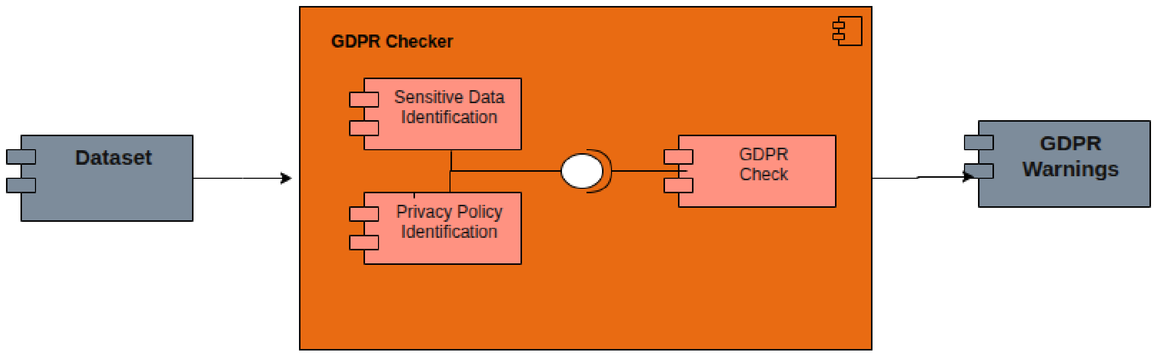


Figure 60: GDPR Checker High Level Architecture

7.3.7 Technology Background

The GDPR Checker is implemented in **Node.js** with **TypeScript** and loads **OWL/ODRL** ontology files at runtime to represent GDPR concepts like data categories and privacy properties. When triggered via a **REST** endpoint, it receives an asset ID, retrieves the associated dataset, and uses a lightweight semantic reasoner to compare column labels against the ontologies. If it finds potential conflicts with the GDPR rules, it generates warnings and mitigation suggestions. By combining Node.js/TypeScript with semantic web libraries, the GDPR Checker remains scalable, extensible, and seamlessly integrated into the PISTIS platform.

7.3.8 Graphical User Interface

The component doesn't have a GUI.

7.3.9 Developer Documentation

The developer documentation for the GDPR Checker component may be accessed on the official PISTIS project documentation site at the following link: https://docs.pistis-market.eu/developers/gdpr-checker/gdpr-checker

7.3.10 Source Code

The source code of the GDPR Checker component is maintained in a private GitHub repository under the PISTIS project, accessible only to the PISTIS consortium and the reviewers:

https://github.com/PISTIS-Platform/besu_ledger-scee-scc-gdpr-checker/tree/main/node-server/src/checkers/GDPRChecker

While the GDPR Checker is conceptually distinct, it is technically integrated within the same Node.js server as the Smart Contract Execution Engine (SCEE) and the Smart Contract Checker, all hosted in the same repository. Additionally, the repository includes the Data Ledger implementation, developed using the Hyperledger Besu Quorum framework, providing a unified solution for these interconnected components. The attached link leads to the GDPR Checker's implementation logic within this comprehensive repository.

7.4 Searchable Encryption

7.4.1 Component Description

The Searchable Encryption component is designed to provide secure and privacy-preserving search capabilities within the PISTIS platform. It enables authorized users to perform searches over encrypted data without exposing the actual data content but only to authenticated buyers featuring the necessary set of attributes (e.g. a buyer who has the necessary tokens in their wallet or a buyer who has signed a contract for this type of data).

This component supports two main types of queries: 1) queries over metadata (keywords) associated to encrypted pointers that interpoint to the encrypted datasets (metadata based Searchable Encryption) and 2) more granular queries over specific data fields (rows, columns) comprising tabular datasets. The latter also leverages metadata based Searchable Encryption capabilities, where metadata are keywords associated to the data fields of interest. The metadata queries allow efficient filtering and discovery of relevant datasets, while the second modality enables more granular, content-based searches within encrypted datasets using advanced cryptographic techniques.

By combining these two querying methods, the Searchable Encryption component balances usability and security, allowing users to find and access data assets they are authorized for without compromising data privacy. The component integrates with PISTIS identity and access management systems to ensure that only authenticated and authorized users can perform searches, maintaining strict control over data access.

7.4.2 Main Improvements in Beta Version

The Searchable Encryption component is introduced for the first time in the Beta version of the PISTIS platform. It provides a novel capability to perform secure, privacy-preserving searches over encrypted datasets and their metadata.

7.4.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
US_01	Data Consumer	search datasets based on specific keywords	I can buy them	Beta	Search results	Done	PISTIS.OUS.9
US_02	Data Provider	hide actual type of data	I can protect my privacy	Beta	Search results	Done	PISTIS.OUS.9
US_03	Data provider	only users with specific attributes to search my data	I can protect my privacy	Beta	Search results	Done	PISTIS.OUS.9
US_04	Data Provider	to dynamically update the encrypted keyword	the keyword to be more accurate upon	Beta	Search results	Done	PISTIS.OUS.9

			changes				
US_05	PISTIS Administrator	be sure that only authorised PISTIS user can search and buy datasets	confidentiality and authentication requirements are met	Beta	Search results	Done	PISTIS.OUS.9 & PISTIS.SOUS.01
US_06	PISTIS Administrator	be sure that a reasonable number of queries per user will be done	the PISTIS platform to be efficient	Beta	Search results	Done	PISTIS.OUS.9 & PISTIS.SOUS.01

7.4.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	Searchable Encryption supports encryption with specific attributes defined by the Data Provider.	UC_3, UC_4, UC_5	These attributes are part of the verifiable credentials of Data Consumers.
FR_02	Searchable Encryption supports search upon encrypted data with specific attributes defined by the Data Provider.	UC_1, UC_2, UC_5	These attributes are part of the verifiable credentials of Data Consumers.

7.4.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Functional Suitability	NFR1	All entities with the specified attributes will be in position to decrypt the corresponding encrypted data (e.g., index).
Performance efficiency	NFR2	Encryption/decryption and search functionalities should be performed in efficient way.
Compatibility	NFR3	All entities with the specified attributes will be in position to decrypt the corresponding encrypted data (e.g., index).
Usability	NFR4	Authorized entities should easily search encrypted data indexes through the PISTIS platform.
Reliability	NFR5	Search results should be reliable and displayed only to authorised entities.
Security	NFR6	Data confidentiality and privacy of data provider and data consumer should be supported.

7.4.6 Component Architecture

Figure 61 below presents the high-level architecture of the Searchable Encryption including the internal components of the tool. Such a component represents a significant advancement in secure data handling. It exemplifies how modern encryption techniques can be harmonized with emerging technologies like blockchain to create a secure, transparent, and efficient data management system. This component is pivotal in enabling the PISTIS platform to handle sensitive data with the utmost security while ensuring that the data remains accessible and useful for authorized users.

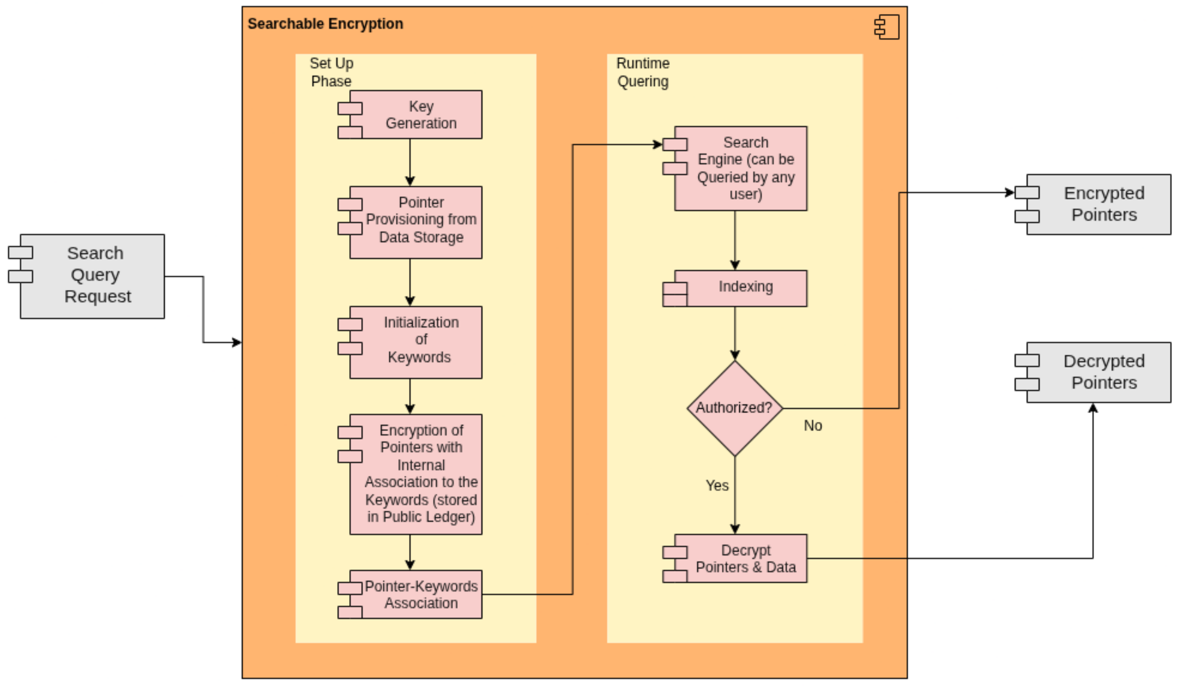

Searchable Encryption
Set Up Phase
Key Generation
Pointer Provisioning from Data Storage
Initialization of Keywords
Encryption of Pointers with Internal Association to the Keywords (stored in Public Ledger)
Pointer-Keywords Association
Runtime Quering
Search Engine (can be Queried by any user)
Indexing
Authorized?
No
Yes
Decrypt Pointers & Data
Search Query Request
Encrypted Pointers
Decrypted Pointers

Figure 61: Searchable Encryption high-level Architecture

7.4.7 Technology Background

The Searchable Encryption component is designed around the concept of **Dynamic Symmetric Searchable Encryption (DSSE)**, a system that allows for the searching of encrypted data quickly and efficiently. DSSE is crucial for maintaining privacy while ensuring that data retrieval processes are swift and effective, especially important when dealing with large volumes of data stored in cloud environments.

This component is implemented using Node.js and the implementation also involves the use of cryptographic libraries and APIs that support the operations of DSSE. These libraries ensure the secure handling of encryption keys and the execution of encryption algorithms, maintaining data confidentiality while enabling the search functionality. The use of these technologies ensures that the Searchable Encryption component can securely manage, index, and retrieve encrypted data without exposing sensitive information.

7.4.8 Graphical User Interface

As this component is a backend component, no UI is provided.

7.4.9 Developer Documentation

The developer documentation for the Searchable Encryption component may be accessed on the official PISTIS project documentation site at the following link:

https://docs.pistis-market.eu/developers/searchable-encryption/searchable-encryption

7.4.10 Source Code

The source code of the Searchable Encryption component is maintained in a private GitHub repository under the PISTIS project, accessible only to the PISTIS consortium and the reviewers at https://github.com/PISTIS-Platform/searchable-encryption.

7.5 Encryption/Decryption Engine

7.5.1 Component Description

The Encryption/Decryption Engine in the PISTIS platform is a flexible and essential component that manages data encryption and decryption across the system. It supports common encryption methods to protect data privacy and security.

In the current design, the actual encryption and decryption operations are performed by the OpenDSU client deployed within each factory. Meanwhile, a separate Flask-based server handles the exchange of encryption keys and coordinates key management. Although these parts are technically separate, they function together as a single logical component to ensure secure data handling.

Additionally, the encryption keys used are linked to the user's Self-Sovereign Identity (SSI), reinforcing user control and trust in the data protection process.

This architecture allows for strong security while supporting distributed and factory-specific encryption needs across the platform.

7.5.2 Main Improvements in Beta Version

In the Beta version, the Encryption/Decryption Engine introduces significant enhancements focused on improving security and modularity. The core encryption and decryption processes have been moved to the OpenDSU client running within each factory, enabling more localized and secure data handling. This change reduces the risk of key exposure and improves performance by decentralizing cryptographic operations. Additionally, the Flask-based server now primarily handles the secure exchange and management of encryption keys, allowing for better separation of concerns and easier maintenance. The integration between the OpenDSU clients and the key exchange server is streamlined to support the Self-Sovereign Identity (SSI) framework, ensuring that encryption keys are securely linked to user identities. These improvements collectively strengthen the platform’s ability to protect data confidentiality while supporting flexible and distributed deployments across multiple factories.

7.5.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
US_01	Data Consumer	be able to decrypt encrypted datasets	I can see the actual data	Alpha	Decryption process	Done	PISTIS. OUS.1 0
US_02	Data Provider	be able to encrypt datasets	I can protect data confidentiality	Alpha	Encryption process	Done	PISTIS. OUS.1 0
US_03	User	make sure that my encryption keys are associated with my Self-Sovereign Identity	my data privacy and control are ensured.	Beta	Encryption/D ecryption Process	Done	PISTIS. OUS.1 0

7.5.4 Functional Requirements

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	PISTIS should support data encryption on data transactions.	UC_1, UC_2	The receiver should be able to decrypt the encrypted transaction.

7.5.5 Non-Functional Requirements

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Performance efficiency	NFR1	Encryption/decryption should be performed in efficient way.
Compatibility	NFR2	The data receiver should be able to decrypt the transferred data.
Reliability	NFR3	No other parties should be able to decrypt and read the actual data.
Security	NFR4	Only the data receiver should be in position to decrypt the data transaction for confidentiality purposes.
Portability	NFR5	Upon an expired certificate the new one should be used.

7.5.6 COMPONENT ARCHITECTURE

The Encryption/Decryption Engine is designed to secure and retrieve data by transforming it to and from an encrypted format, ensuring data privacy and integrity. This process is essential for protecting sensitive information in various applications. The engine includes two primary components: the 'Encrypt' module and the 'Decrypt' module. The Encrypt module takes an unencrypted dataset, often referred to as plaintext, and uses cryptographic algorithms to convert it into an encrypted form, known as ciphertext. This encrypted data ensures that sensitive information remains secure during storage or transmission. Conversely, the Decrypt module performs the reverse operation. It takes the encrypted dataset and applies the corresponding decryption algorithms to convert it back to its original form, making the data accessible for authorized use. The entire process is tightly integrated with the Self-Sovereign Identity (SSI) of the user, which manages the encryption keys, thereby ensuring that only the user with the correct identity credentials can decrypt the data. This mechanism is pivotal in maintaining data confidentiality and access control in compliance with privacy regulations.

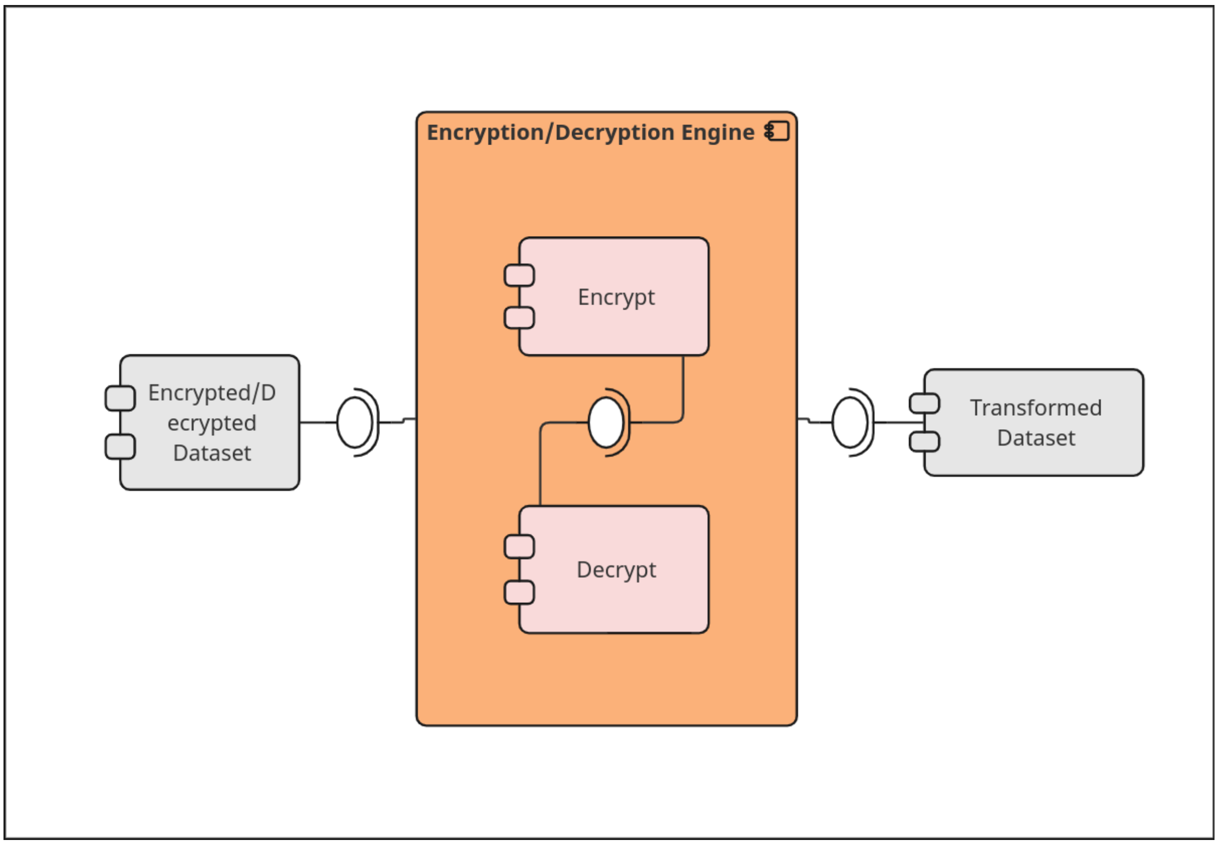


Figure 62: Encryption/Decryption Engine High Level Architecture

7.5.7 TECHNOLOGY BACKGROUND

The Encryption/Decryption Engine component combines **Flask**, a lightweight **Python** web framework, with high-performance **C** code to handle its core cryptographic functions. The encryption and decryption logic is implemented in C to leverage its speed and direct hardware access, essential for efficient and secure processing of cryptographic operations. Flask serves as the interface layer, managing communication and coordination, including key exchange. The component also integrates with **EJBCA**, a robust Public Key Infrastructure (PKI) software that manages digital certificates and public keys. EJBCA supports multiple certificate authorities and hierarchical CA levels, allowing the engine to maintain a flexible and comprehensive security framework within a single deployment. In the Beta version, significant architectural changes have been introduced: the **OpenDSU** client now performs encryption and decryption locally within each factory, while the Flask server focuses on secure key management and exchange. This separation improves security by limiting key exposure and enhances scalability for distributed factory environments. Together, these technologies create a strong, modular, and extensible encryption system aligned with Self-Sovereign Identity principles.

7.5.8 Graphical User Interface

As this component is a backend component, no UI is provided.

7.5.9 Developer Documentation

The developer documentation for the Encryption/Decryption Engine component may be accessed on the official PISTIS project documentation site at the following link:

https://docs.pistis-market.eu/developers/encryption-decryption-engine/encryption-decryption-engine

7.5.10 Source Code

The source code of the Encryption/Decryption Engine component is maintained in a private GitHub repository under the PISTIS project, accessible only to the PISTIS consortium and the reviewers at https://github.com/PISTIS-Platform/encryption-decryption-engine.

7.6 Access Policy Editor

7.6.1 Component Description

Access Policy Editor is a component integrated with Keycloak[13] identity and access management platform within PISTIS and IAM API. The component serves as a centralized tool allowing PISTIS Data Factory Administrators to define and apply the access policies for the data to be placed over the PISTIS platform.

The primary goal is to simplify the definition of scope-based policies through an intuitive web-based UI editor. By doing so, administrators gain the ability to tailor access controls for specific

[13] https://www.keycloak.org

use cases, encompassing user roles, and access to targeted resources within the PISTIS ecosystem. This functionality ensures that the access policies align with the unique organizational structure and requirements so as data living in PISTIS platform are findable with proper access to other organizations.

Access policies generated by the Keycloak-based Access Policy Editor provide a multifaceted approach to access control. Firstly, they dictate who has the privilege to access distinct PISTIS Organization resources, ranging from specific features within the PISTIS Platform to exclusive datasets owned by the organization. Secondly, these policies define the scope or rights associated with accessible PISTIS Organization resources. This extends beyond conventional permissions, including Create, Read, Update, Delete, and Admin, to incorporate PISTIS-specific policies such as Trading, Transformation, Pricing, and more. Additionally, the editor allows administrators to finely tune access on nested objects or attributes within a specific PISTIS Organization's resource. For example, an administrator can grant read access to an entire data stream while restricting update permissions to a specific attribute, providing granular control over resource accessibility.

Internally, the Access Policy Editor relies on Keycloak's infrastructure. Keycloak employs a secure and scalable database system to store and retrieve the intricate policies defined by administrators. This ensures that policy information is organized, quickly accessible, and securely managed. Furthermore, Keycloak leverages its advanced indexing service to optimize the efficiency of policy enforcement during runtime. The indexing service plays a pivotal role in accelerating the retrieval of policies, contributing to the overall responsiveness and performance of the Access Policy Editor within the PISTIS platform. These internal components work seamlessly to provide a dynamic, responsive, and secure access control mechanism tailored to the specific needs of organizations utilizing the Keycloak-based Access Policy Editor in the PISTIS environment.

Access Policy Editor will provide a web-based GUI to allow PISTIS Organization Admins to manage their organization access and permissions as well as PISTIS Users to define extra access policies during a Data Asset Injection and Publication phases.

7.6.2 Main Improvements in Beta Version

Improvements in beta version are in two directions. In the first direction, a UI-based integration with “Job Configurator” has been added (while in alpha version integration has been accomplished via a REST call, without extra configuration available) and publication policies with Group (Organization) attributes has been supported.

Additionally, beta version is powered by a standalone version on Access Policy Editor, for PISTIS Administrator and Org Administrators, allowing them to manage Groups (Organizations), Users (within a Group/Organization) and finally offering more advanced policy rules. Finally, user management for Org Administrators is planned to be made available in Factory frontend platform.

7.6.3 Component Backlog

This section provides the full set of features that belong to the backlog of the component.

ID #	Use Case			Backlog Priority	Acceptance Criteria	Status	WP1 User Stories
	As a <Role>	I want to <Action>,	so that <Reason>				
UC_01	I want to <Action>,	so that <Reason>	Can control access to the asset from the organization users	Alpha	Access to the asset is governed by the policies created by the asset owner	Done	PISTIS.OUS.05 PISTIS.OUS.01
UC_02	Data Provider	Define access policies on a published asset	Can control access to the asset from outside users	Alpha	Access to the asset is governed by the policies created by the asset owner	Done	PISTIS.OUS.05 PISTIS.OUS.06
UC_03	PISTS Organization Admin	Define the roles and attributes of the organization users	Can create role based and attribute-based access control policies	Alpha	User roles and attributes are stored in the IAM database	Done	PISTIS.OUS.05
UC_04	Data Provider	Create and manage complex access control policies on all organization assets	Can have fine grain control on who and when can access a specific asset	Alpha (first set of supported policies)/ Beta	A user can create arbitrary access policies based on roles and attributes of users/assets/ organizations. Access control policies can also be time based and context based	Done (first batch of supported policies)	PISTIS.OUS.05
UC_05	Data Provider	Define access control policies based on a user's eIDAS credentials and attributes	Can control access to an asset from external eIDAS certified user	Beta	Access to an asset is based on the attributes of a user's eIDAS credentials	Done	PISTIS.OUS.05
UC_06	PISTS Organization Admin	Manage own organization users	Can create and modify user accounts	Beta	Users' account details are stored in IAM database	Done	PISTIS.OUS.05

7.6.4 FUNCTIONAL REQUIREMENTS

This section provides the functional requirements of the component.

ID	Description	Related Use Cases	Comments
FR_01	Allow a PISTIS User (data provider) to create extra access policies, referring to its Organization, for a Data Asset during Injection phase.	PISTIS.OUS.01, PISTIS.OUS.05	
FR_02	Allow a PISTIS User (data provider) to create extra access policies for a Data Asset during Publication phase.	PISTIS.OUS.01, PISTIS.OUS.05	
FR_03	Allow PISTIS Organization Administrators to manage their organization users and assign roles.	PISTIS.OUS.02	
FR_04	Allow PISTIS Organization Administrators to manage access policies for Data Assets belonging to users of the organization.	PISTIS.OUS.05	
FR_05	Provide list of effective Access Policies of a PISTIS Asset to be bundled in the Blockchain.	PISTIS.OUS.01	Provided to Asset Description Bundler and Data CheckIn during Injection and Publication of a Data Asset.
FR_06	Register a new PISTIS Asset with its Access Policies specific attributes in Keycloak and create a set of default policies within the specific Organization.	PISTIS.OUS.01	Provided to Data CheckIn during Asset Injection.
FR_07	Register the publication of a PISTIS Asset with its Access Policies specific attributes in Keycloak and create a set of default policies within PISTIS ecosystem.	PISTIS.OUS.01	Provided to Data CheckIn during Asset Publication.
FR_08	Register a used-defined access policy for a PISTIS Asset during asset injection phase.	PISTIS.OUS.01	Provided during Asset Injection.
FR_09	Register a used-defined access policy for a PISTIS Asset during Publication phase.	PISTIS.OUS.01	Provided during Asset Publication.
FR_10	Management of complex access policies, defined by PISTIS Organization Administrators via Access Policy Editor web-based.	PISTIS.OUS.05	Provided to Access Policy Editor.

7.6.5 NON-FUNCTIONAL REQUIREMENTS

The following table presents the non-functional requirements of the component (if any).

Requirement Sub-category	Id	Description (Detailed description of the requirement)
Functional Suitability	NFR1	The access policy editor complies with all specified functional requirements.
Performance efficiency	NFR2	The defined access policies can be enforced, by the access policy engine, without introducing long delays to the rest of the operations.

Compatibility	NFR3	The access policy editor REST API service is compatible with all other components capable of sending REST API requests and processing the received responses.
Usability	NFR4	The definition of the access policies is simplified to the point that an untrained user can select the correct policies without having software development experience.
Security	NFR5	The access policy editor can be accessed only by authorized users.
Security	NFR6	Definition of access policies for specific assets is done only by users that belong in the same organization.

7.6.6 Component Architecture

The main elements of Access Policy Editor are:

- The API Adapter will provide the necessary integration layer with IAM API so as execute complex operations within Keycloak get executed and therefore leverage user's experience as well as component's efficacy. This API mainly servers Access Policy Editor's operations and certain operation for factory management.
- An included in the PISTIS UI web-based GUI to allow a PISTIS User (data provider) to specify extra access policies (restrictions) during a Data Asset Injection or Publication.
- An included in the PISTIS UI web-based GUI to allow a PISTIS Organization Administrator to manage his organization users.
- A standalone Web-based GUI to provide to the PISTIS Organisation Administrators with ease management of Users and Policies within their organization.

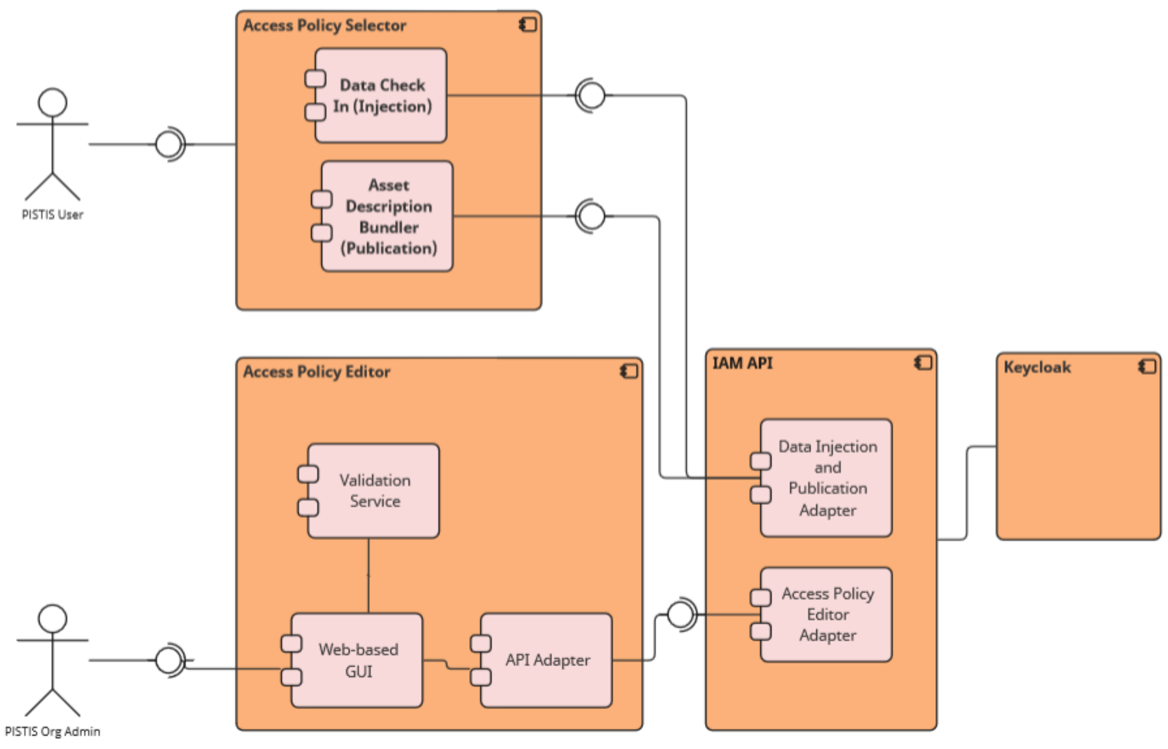

Access Policy Selector
Data Check In (Injection)
Asset Description Bundler (Publication)
PISTIS User
Access Policy Editor
Validation Service
Web-based GUI
API Adapter
PISTIS Org Admin
IAM API
Data Injection and Publication Adapter
Access Policy Editor Adapter
Keycloak

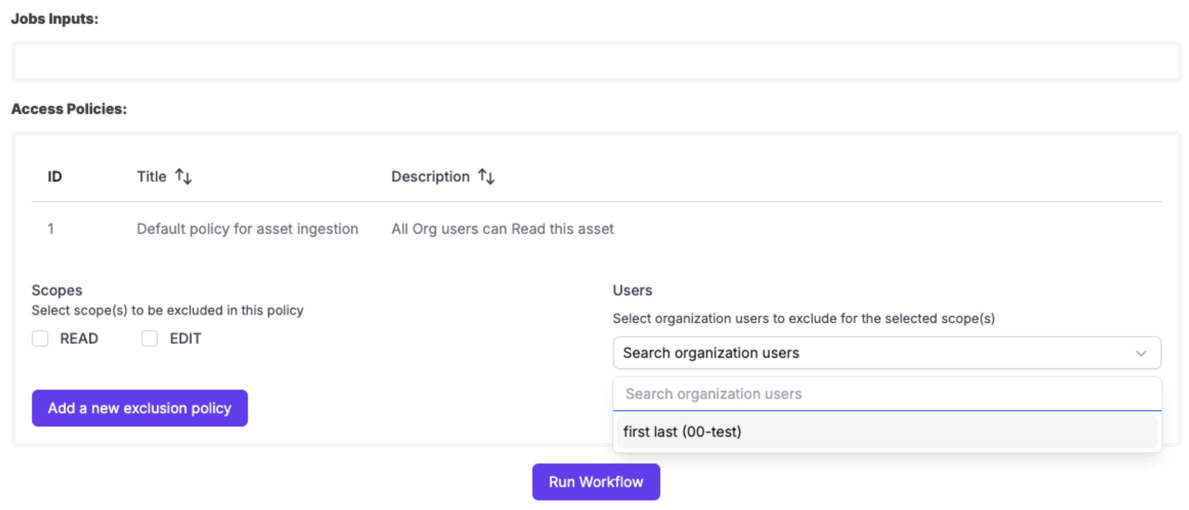
Figure 63: Access Policy Editor Internal Architecture

7.6.7 Technology Background

PISTIS Access Policy Editor component, is a web-based GUI realized with NuxtJS[14] and strongly relies on developed Identity and Access Management APIs that subsequently rely on Keycloak APIs. The PISTIS-centric REST API is developed using Java technology and specifically Spring Boot framework version 3.x.

7.6.8 Graphical User Interface

During Data Ingestion a PISTIS User is offered to create exclusion access policies for users that belong to his organization. This part of Access Policy Editor UI is part of PISTIS Factory UI and a screenshot is provided in Figure 64. In this figure, the default access policy that is applied upon asset ingestion is displayed together with the offering functionality of the component. The default access policy applied during asset ingestion claims that all users belonging to creator's PISTIS Organization are able to read and edit this asset.



Jobs Inputs:

Access Policies:

ID	Title	Description
1	Default policy for asset ingestion	All Org users can Read this asset

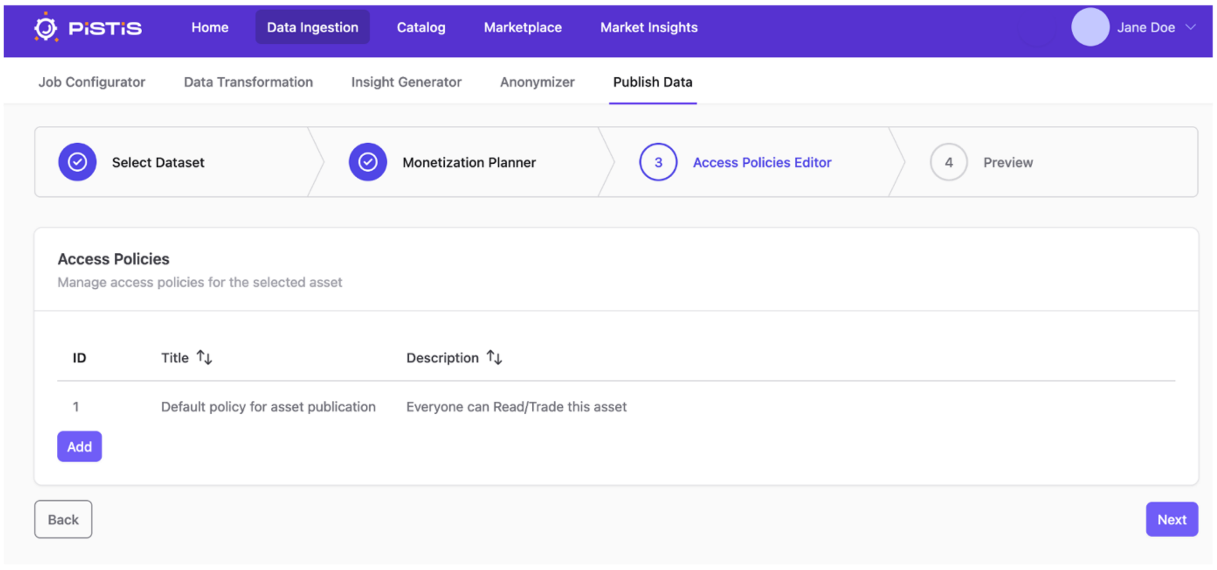
Scopes
Select scope(s) to be excluded in this policy
☐ READ ☐ EDIT

Users
Select organization users to exclude for the selected scope(s)
Search organization users
first last (00-test)

Figure 64: Listing of access policies during data asset injection phase and definition of new exclusion access policy for organization's users

During Data Publication a PISTIS User is offered to create exclusion access policies with several criteria, as presented below. This part of Access Policy Editor UI is part of PISTIS Factory UI and relevant screenshots are provided in Figure 65 and Figure 66. In details, the default access policy that is applied upon asset publication is displayed together with the offering functionalities of the component. The default access policy applied during asset publication claims that all users of PISTIS Organization can read and trade this asset. Moreover, the offered criteria for generation exclusion policies include PISTIS Organizations and their attributes, namely the type, the origin (country), the domain of activity and the size of a PISTIS Organization, as illustrated in Figure 66.

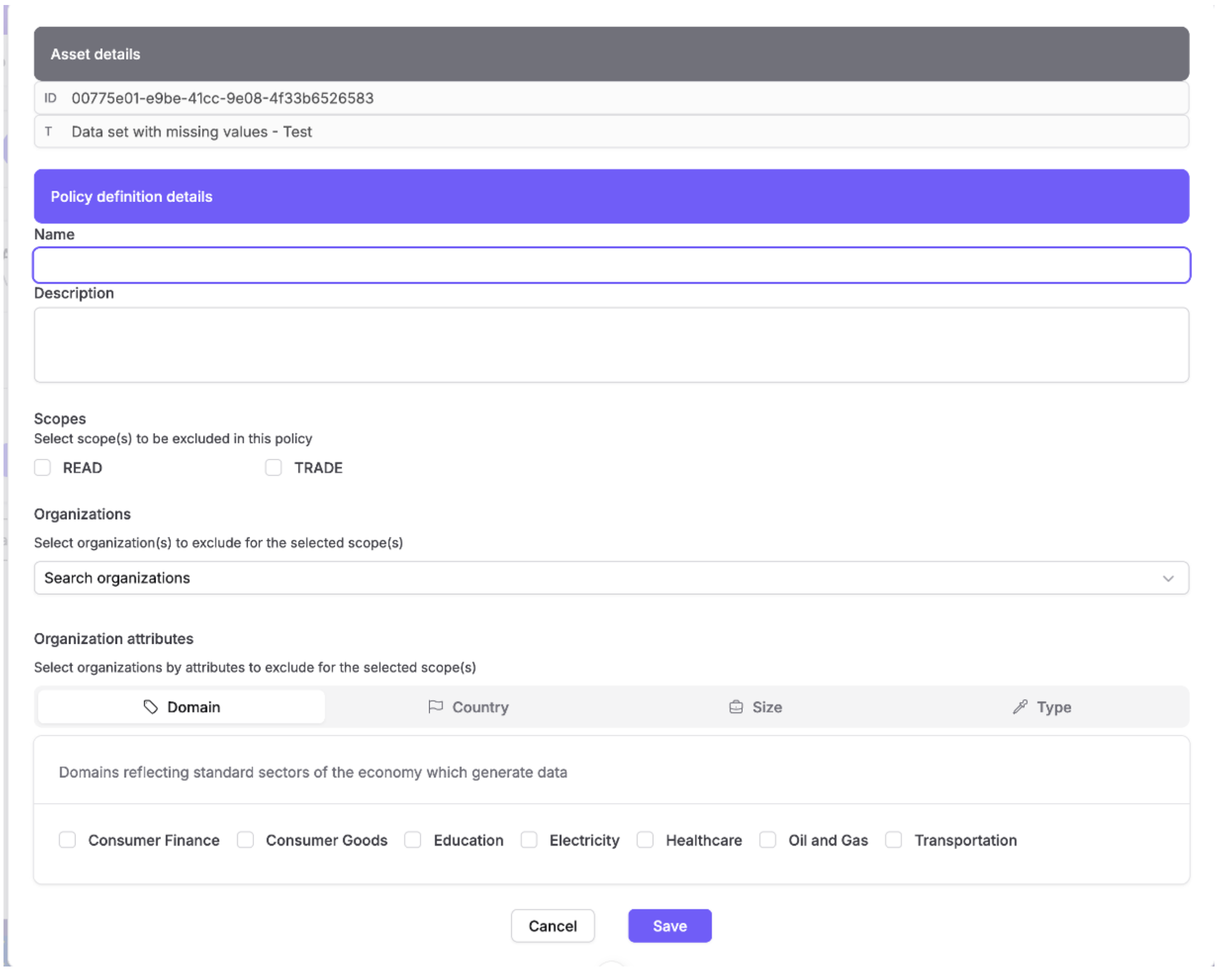
[14] https://nextjs.org



Access Policies
Manage access policies for the selected asset

ID	Title	Description
1	Default policy for asset publication	Everyone can Read/Trade this asset

Figure 65: Listing of access policies during data asset publication phase (the default access policy)


Asset details

ID 00775e01-e9be-41cc-9e08-4f33b6526583
T Data set with missing values - Test

Policy definition details

Name:

Description:

Scopes
Select scope(s) to be excluded in this policy
READ TRADE

Organizations
Select organization(s) to exclude for the selected scope(s)
Search organizations

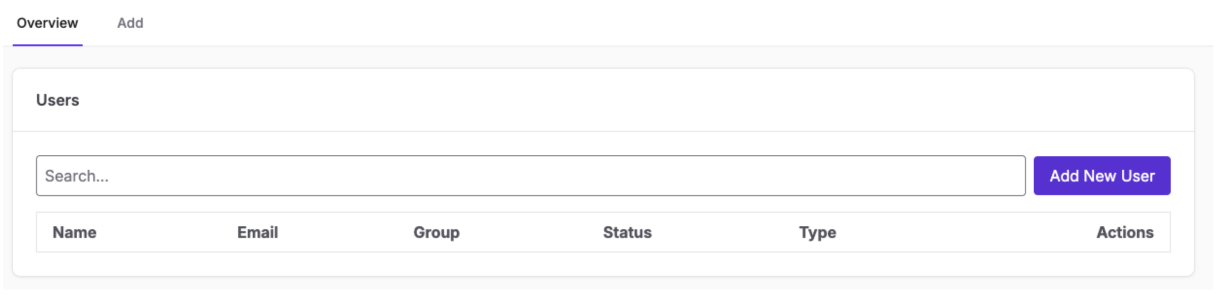
Organization attributes
Select organizations by attributes to exclude for the selected scope(s)

Domains reflecting standard sectors of the economy which generate data

Consumer Finance, Consumer Goods, Education, Electricity, Healthcare, Oil and Gas, Transportation

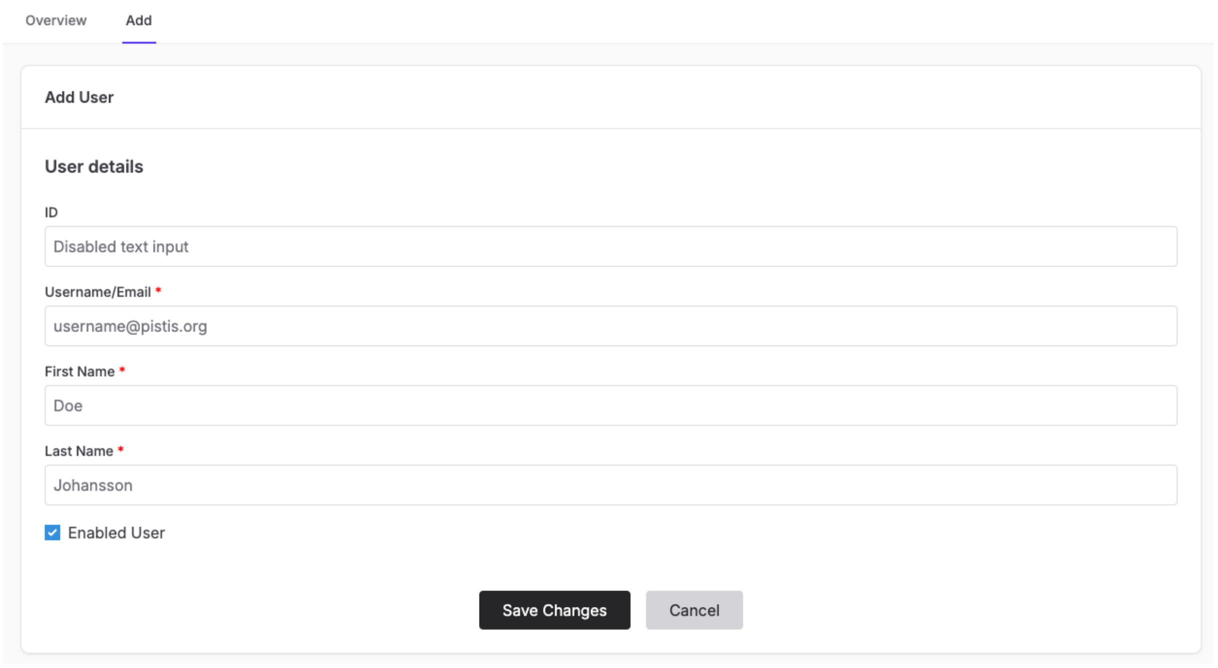
Figure 66: Registration of a new access policy during data asset publication phase

Another part of the Access Policy component, also included in the PISTIS UI web-based GUI, aims to allow a PISTIS Organization Administrator to manage his organization users. Figure 67 and Figure 68 illustrate this functionality.



Name	Email	Group	Status	Type	Actions

Figure 67: Listing of PISTIS users within access policy editor

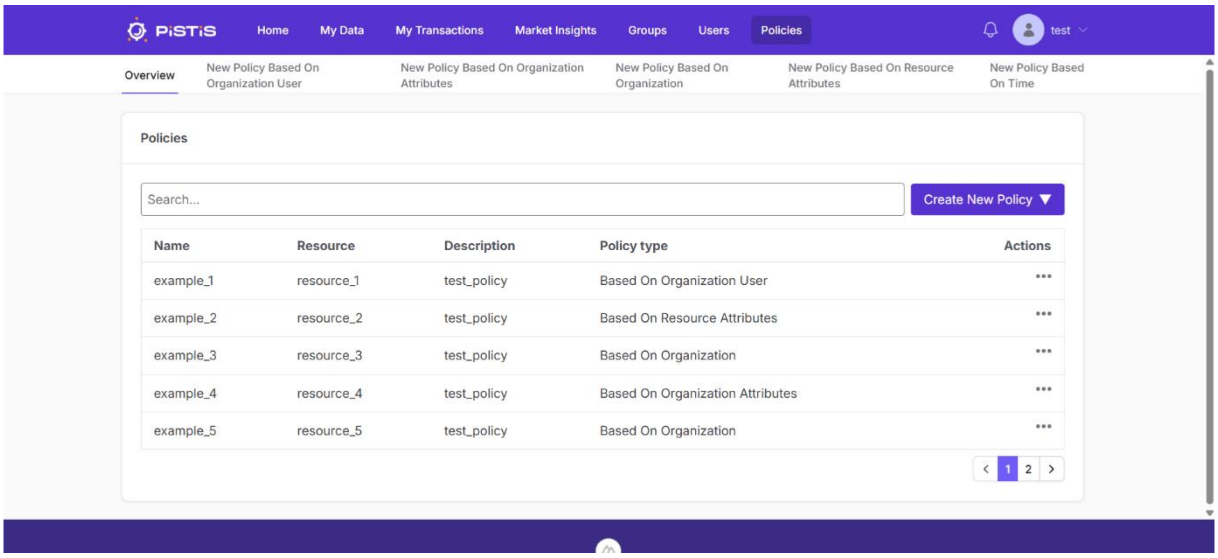


ID	Disabled text input
Username/Email *	username@pistis.org
First Name *	Doe
Last Name *	Johansson
✓ Enabled User	

Figure 68: Create (or edit) a PISTIS user

Finally, Access Policy Editor component offers a standalone web-based GUI version to allow PISTIS Organization Administrators to manage access policies for the registered (published) assets. The access policies generated again follow the exclusion logic and address several criteria, as presented in the following screenshots.

The screenshot below lists all the registered access policies within the access policy editor. The following attributes of each access policy are shown in the table: the name, the affected resource, the description, and the type. Each policy can be edited or deleted (under the "Actions" column). Access policies are shown paginated. Additionally, search functionality is provided within the list of access policies. Finally, the option to create a new access policy is available.

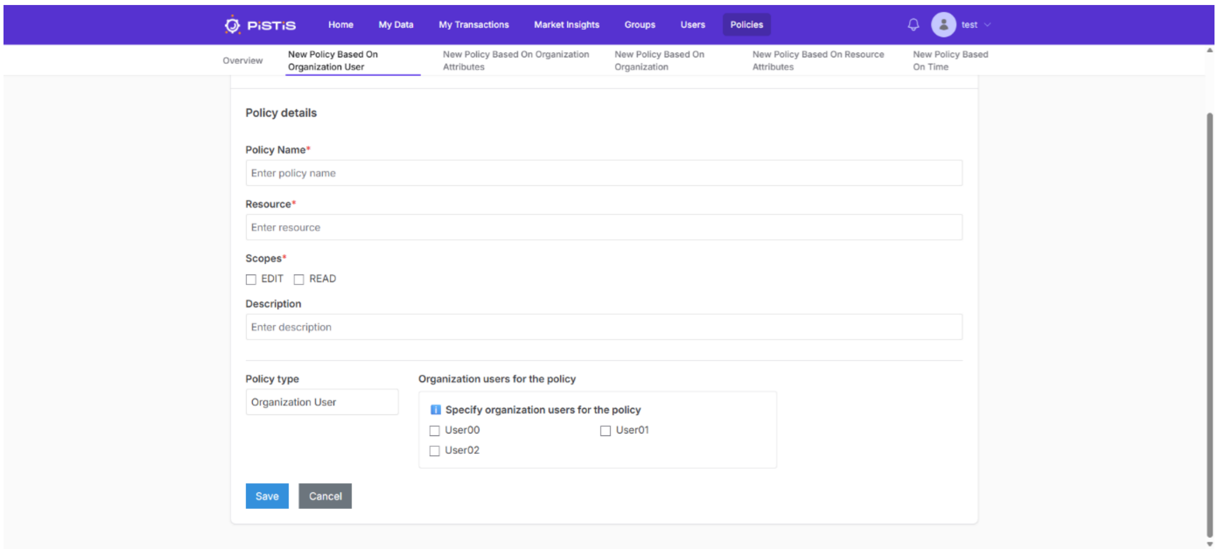

PiSTiS
Home
My Data
My Transactions
Market Insights
Groups
Users
Policies
test
Overview
New Policy Based On Organization User
New Policy Based On Organization Attributes
New Policy Based On Organization
New Policy Based On Resource Attributes
New Policy Based On Time
Search...
Create New Policy
1
2

Policies

Name	Resource	Description	Policy type	Actions
example_1	resource_1	test_policy	Based On Organization User	•••
example_2	resource_2	test_policy	Based On Resource Attributes	•••
example_3	resource_3	test_policy	Based On Organization	•••
example_4	resource_4	test_policy	Based On Organization Attributes	•••
example_5	resource_5	test_policy	Based On Organization	•••

Figure 69: Listing of registered access policies within the access policy editor

The screenshot below shows the creation of a new access policy for a user of an organization. The registration of such an access policy can be done by both the PISTIS administrator and the organization's administrator.


PiSTiS
Home
My Data
My Transactions
Market Insights
Groups
Users
Policies
test
Overview
New Policy Based On Organization User
New Policy Based On Organization Attributes
New Policy Based On Organization
New Policy Based On Resource Attributes
New Policy Based On Time
Save
Cancel

Policy details

Policy Name*
Enter policy name

Resource*
Enter resource

Scopes*
☐ EDIT ☐ READ

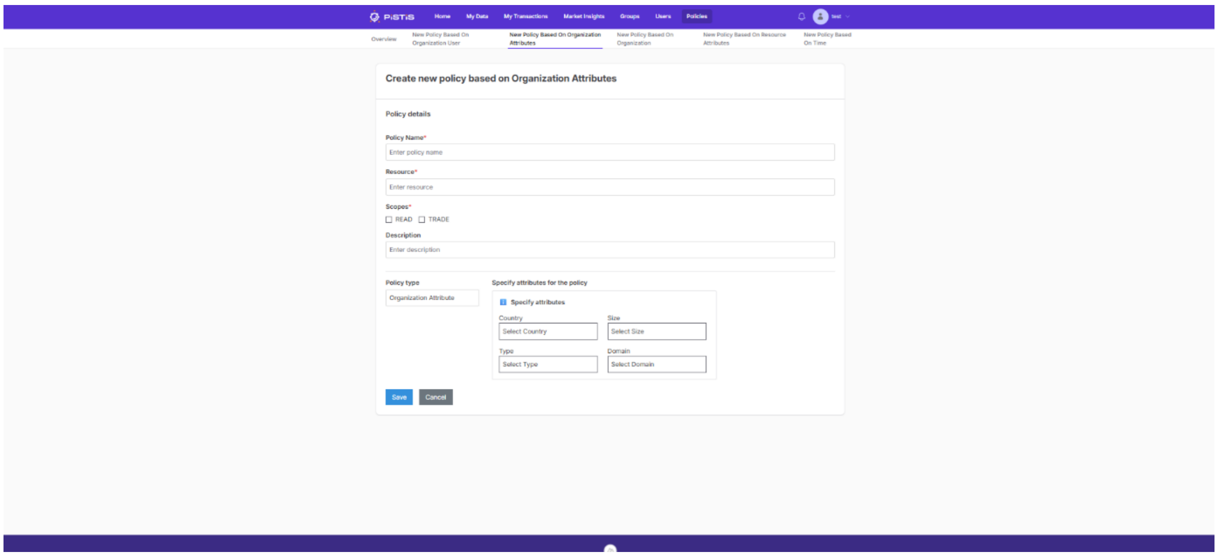
Description
Enter description

Policy type
Organization User

Organization users for the policy
Specify organization users for the policy
☐ User00 ☐ User01
☐ User02

Figure 70: Registration of a new access policy within access policy editor based on organization user

The screenshot below shows the creation of a new access policy based on the attributes of a user's organization, which are: the country, the size, the type, and the domain of the organization. Such a policy can be added by both the PISTIS administrator and the organization's administrator.


PiSTiS
Home
My Data
My Transactions
Market Insights
Groups
Users
Policies
test
Overview
New Policy Based On Organization User
New Policy Based On Organization Attributes
New Policy Based On Organization
New Policy Based On Resource Attributes
New Policy Based On Time
Save
Cancel

Create new policy based on Organization Attributes

Policy details

Policy Name*
Enter policy name

Resource*
Enter resource

Scopes*
☐ READ ☐ TRADE

Description
Enter description

Policy type
Organization Attribute

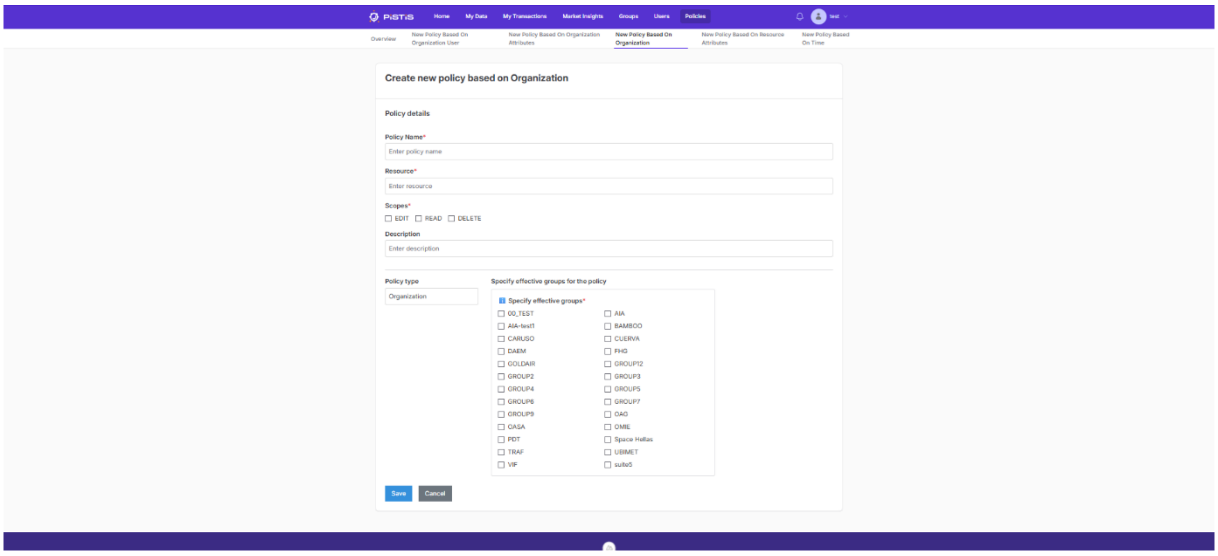
Specify attributes for the policy

Specify attributes

Country	Size
Select Country	Select Size
Type	Domain
Select Type	Select Domain

Figure 71: Registration of a new access policy within access policy editor based on user's organization attributes

The screenshot below shows the creation of a new access policy for an organization. The registration of such an access policy can only be done by the PISTIS administrator.


PiSTiS
Home
My Data
My Transactions
Market Insights
Groups
Users
Policies
test
Overview
New Policy Based On Organization User
New Policy Based On Organization Attributes
New Policy Based On Organization
New Policy Based On Resource Attributes
New Policy Based On Time
Save
Cancel

Create new policy based on Organization

Policy details

Policy Name*
Enter policy name

Resource*
Enter resource

Scopes*
☐ EDIT ☐ READ ☐ DELETE

Description
Enter description

Policy type
Organization

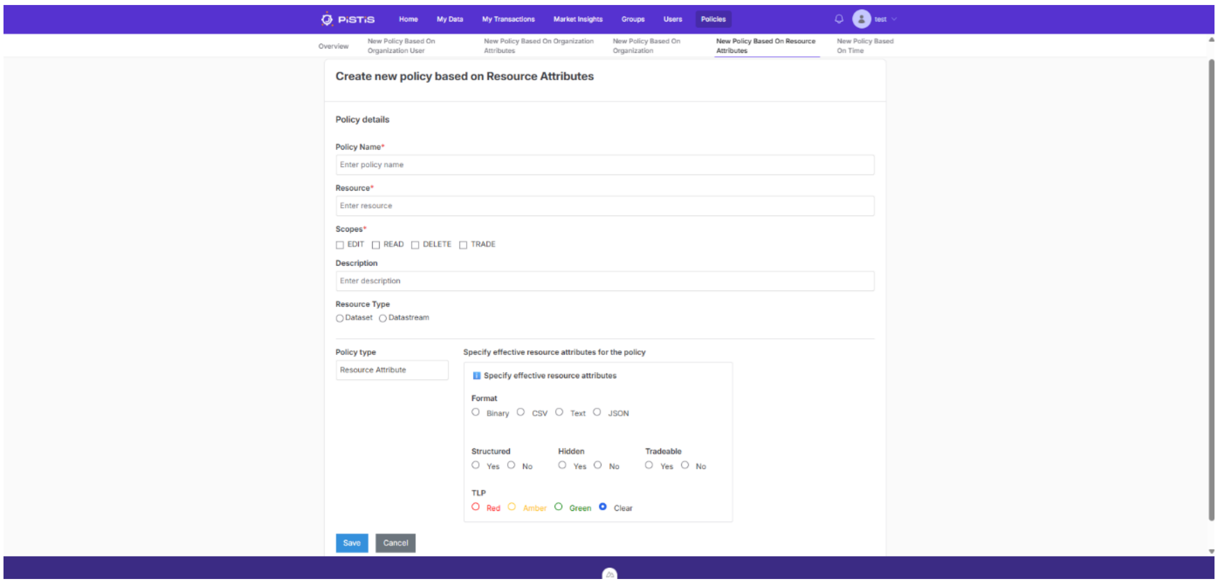
Specify effective groups for the policy

Specify effective groups*

☐ 00_TEST	☐ AIA
☐ AIA-test1	☐ BAMBOO
☐ CARUSO	☐ CUERVA
☐ DAEM	☐ FHG
☐ GOLDAIR	☐ GROUP12
☐ GROUP2	☐ GROUP3
☐ GROUP4	☐ GROUP5
☐ GROUP6	☐ GROUP7
☐ GROUP9	☐ OAG
☐ OASA	☐ OMIE
☐ PDT	☐ Space Hellas
☐ TRAF	☐ UBIMET
☐ VIF	☐ suite5

Figure 72: Registration of a new access policy within access policy editor based on user's organization

The screenshot below shows the creation of a new access policy based on the attributes of an asset (if it is structured, hidden, tradeable, its format, TLP, etc.). This can only be done by the PISTIS administrator.


PiSTiS
Home
My Data
My Transactions
Market Insights
Groups
Users
Policies
test
Overview
New Policy Based On Organization User
New Policy Based On Organization Attributes
New Policy Based On Organization
New Policy Based On Resource Attributes
New Policy Based On Time
Save
Cancel

Create new policy based on Resource Attributes

Policy details

Policy Name*
Enter policy name

Resource*
Enter resource

Scopes*
☐ EDIT ☐ READ ☐ DELETE ☐ TRADE

Description
Enter description

Resource Type
○ Dataset ○ Datastream

Policy type
Resource Attribute

Specify effective resource attributes for the policy

Specify effective resource attributes

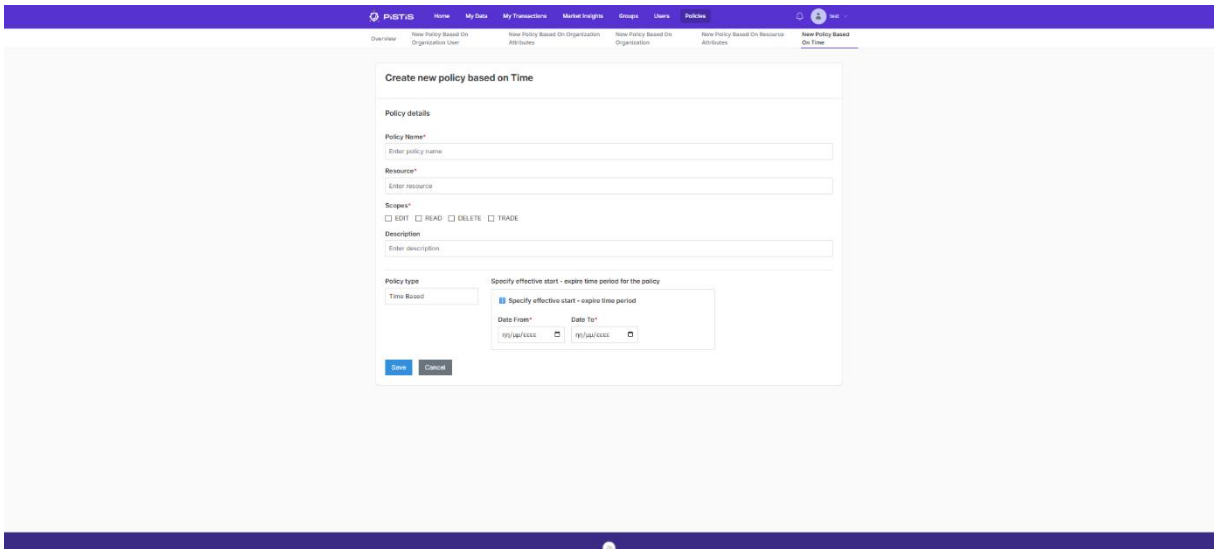
Format
○ Binary ○ CSV ○ Text ○ JSON

Structured ○ Yes ○ No
Hidden ○ Yes ○ No
Tradeable ○ Yes ○ No

TLP
○ Red ○ Amber ○ Green [✓ Clear]

Figure 73: Registration of a new access policy within access policy editor based on data asset's attributes

The screenshot below shows the registration of a new access policy for a specific time period, that is adjusted by the PISTIS administrator or an organization's administrator.


PiSTiS
Home
My Data
My Transactions
Market Insights
Groups
Users
Policies
test
Overview
New Policy Based On Organization User
New Policy Based On Organization Attributes
New Policy Based On Organization
New Policy Based On Resource Attributes
New Policy Based On Time
Save
Cancel

Create new policy based on Time

Policy details

Policy Name*
Enter policy name

Resource*
Enter resource

Scopes*
☐ EDIT ☐ READ ☐ DELETE ☐ TRADE

Description
Enter description

Policy type
Time Based

Specify effective start - expire time period for the policy

Specify effective start - expire time period

Date From*
ηη/μμ/εεεε

Date To*
ηη/μμ/εεεε

Figure 74: Registration of a new access policy within access policy editor based on time period

7.6.9 Developer Documentation

Developer documentation regarding Keycloak integration can be found in

https://docs.pistis-market.eu/developers/keycloak/introduction.

The IAM API Swagger can be found in

https://pistis-market.eu/srv/identity-access-management/api/docs

and can be visualized via

https://pistis-market.eu/srv/identity-access-management/api/docs/swagger-ui/index.html[15]

7.6.10 Source Code

The component's UI that are part of the platform-frontend repository and their code are maintained in GitHub under the PISTIS project and can be accessed via the respective link:

https://github.com/PISTIS-Platform/platform-frontend

Code for the component's standalone version is maintained in GitHub under the PISTIS project and can be accessed via the respective link:

https://github.com/PISTIS-Platform/access-policy-editor

Code for the component's middleware (API) is maintained in GitHub under the PISTIS project and can be accessed via the respective link:

https://github.com/PISTIS-Platform/iam/tree/develop

[15] Paste in the textbox the link: https://pistis-market.eu/srv/identity-access-management/api/docs

8 CONCLUSIONS

The document presents the design and development of the Beta release of the PISTIS components dealing with data discovery, management and protection. It is based on the Alpha version of the components presented in D2.2 and improved according their development backlog.

The functionalities of each component have been described and the technologies that have been exploited for the development of the corresponding alpha releases have been presented. The user stories for each component have been specified and through them the functional and non-functional requirements of the different components have been defined. Moreover, the internal architecture of each component has been provided, showcasing the interconnections among the subcomponents of each component. For the components that have a user interface, respective screenshots depicting the components' functionalities have been also presented.

The design and development of the Beta releases of the PISTIS components presented in this document and in D3.3 will drive the integration activities of the project in the next period towards realizing the delivery of the Beta version of the PISTIS Platform (D4.2). The components will continue to evolve with updates addressing the issues identified during the Beta version evaluation of PISTIS in WP5 and will be encapsulated gradually in the following version 1.0 release of the components and will be documented in the corresponding D2.4 deliverable.