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

D2.2 Data Management and Protection services - Alpha version

Editor(s)	Yury Glikman
Lead Beneficiary	FHG
Status	Final
Version	1.00
Due Date	30/04/2024
Delivery Date	31/05/2024
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	D2.2 Data Management and Protection services - Alpha version
Contributor(s)	Suite5, ATOS, FHG, EUT, SPH, UBITECH, ATHENA, ASSENTIAN, ICCS
Reviewer(s)	VIF, SUITE5, GOLDAIR
Abstract	This deliverable presents the Alpha 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 Alpha 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 Alpha version of the design and implementation of the components from the **Data Monetisation, Transaction Services and Ledgers bundles** is presented in the deliverable D3.2. The source code of the components presented in D2.2 and D2.3 is managed in the project’s GitHub repository at: https://github.com/orgs/PISTIS-Platform/. The access to it can be provided upon a request.

As the next step, the consortium will integrate the components presented in this deliverable and in the deliverable D3.2 into the Alpha release of the PISTIS platform (D4.2).

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 follows the agile development methodology by designing, implementing and evaluating the components in two iterations. At the previous steps of work the technical requirements and user stories were collected (D1.2) and the relevant methods technologies, models and specifications for addressing them were analysed (D2.1, D3.1). Then the initial architecture of the PISTIS Platform depicting the individual components and relations between them was defined in the deliverable D4.1.

This deliverable is the very first (Alpha) version of the design and implementation of the core PISTIS factory components dealing with data discovery, management and protection. The document accompanies the implementation of the components by documenting their purpose, requirements, architecture, technologies used in their implementation and the user interface (if available). 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.

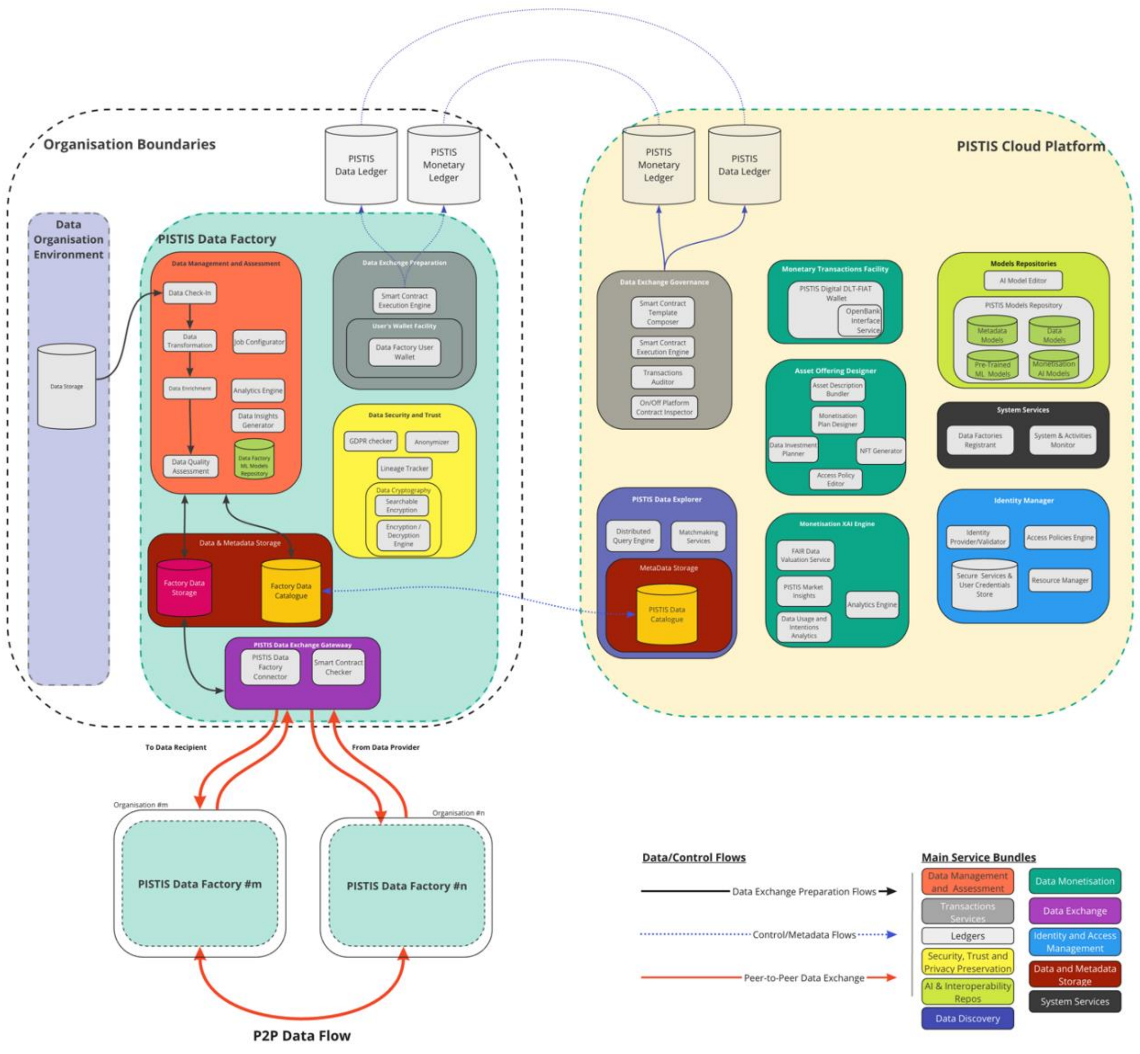


Figure 1: PISTIS Architecture

The Alpha version of another set of the PISTIS components belonging to the **Data Monetisation, Transaction Services and Ledgers bundles** is presented in the deliverable D3.2.

Both sets of the PISTIS components are going to be integrated in the Alpha release of the PISTIS platform (D4.1). This will be the end of the first design & development iteration, which will be followed by two more (Beta and 1.0) iterations of the development process. At the end of each iteration a new version of the platform together with the updated architecture and relevant documentation will be published.

The source code of the alpha releases of the PISTIS components is managed in the project's GitHub repository, access to which can be provided upon a 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 3 - 7 describe the different bundles and present details on the different components belonging to each bundle, as well as the APIs and the Sequence Diagrams of each component.

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

2.1.1 Component Description

Data Check-in will enable 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).

Data Check-In will serve 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
- KAFKA
- MQTT

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 else Data Space Conectors. 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.
- *GetDataFromSubscription*: The possibility to subscribe as a client to a topic is also being evaluated, allowing to get data from this kind of data source (I.e. kafka or MQTT topics). 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 Technology Background

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

Data Check-In will define several Nifi workflows to support the different ways of data ingestion.

2.1.3 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.	Upcoming	PISTIS.OUS.01
UC_03	Data Provider	Inject Data coming from subscription (kafka, MQTT)	Have data available in Pistis ecosystem	Beta	Data stored properly in Factory Data Storage.	Upcoming	PISTIS.OUS.01
UC_04	Data Provider	Inject Data coming from a Data Space (API)	Have data available in Pistis ecosystem	Beta	Data stored properly in Factory Data Storage.	Upcoming	PISTIS.OUS.01
UC_05	Data Provider	Manage Data Check-In from GUI	Have data available in Pistis ecosystem	Beta	Data stored properly in Factory Data Storage.	Upcoming	PISTIS.OUS.01

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

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

2.1.4 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.

2.1.5 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.

2.1.6 Component's Main Elements and Internal 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 2, 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 iwill include dedicated submodules for every kind of ingestion needed by PISTIS, including FTP batch file uploads, Kafka or MQTT listeners for streaming data, and API retrieval.

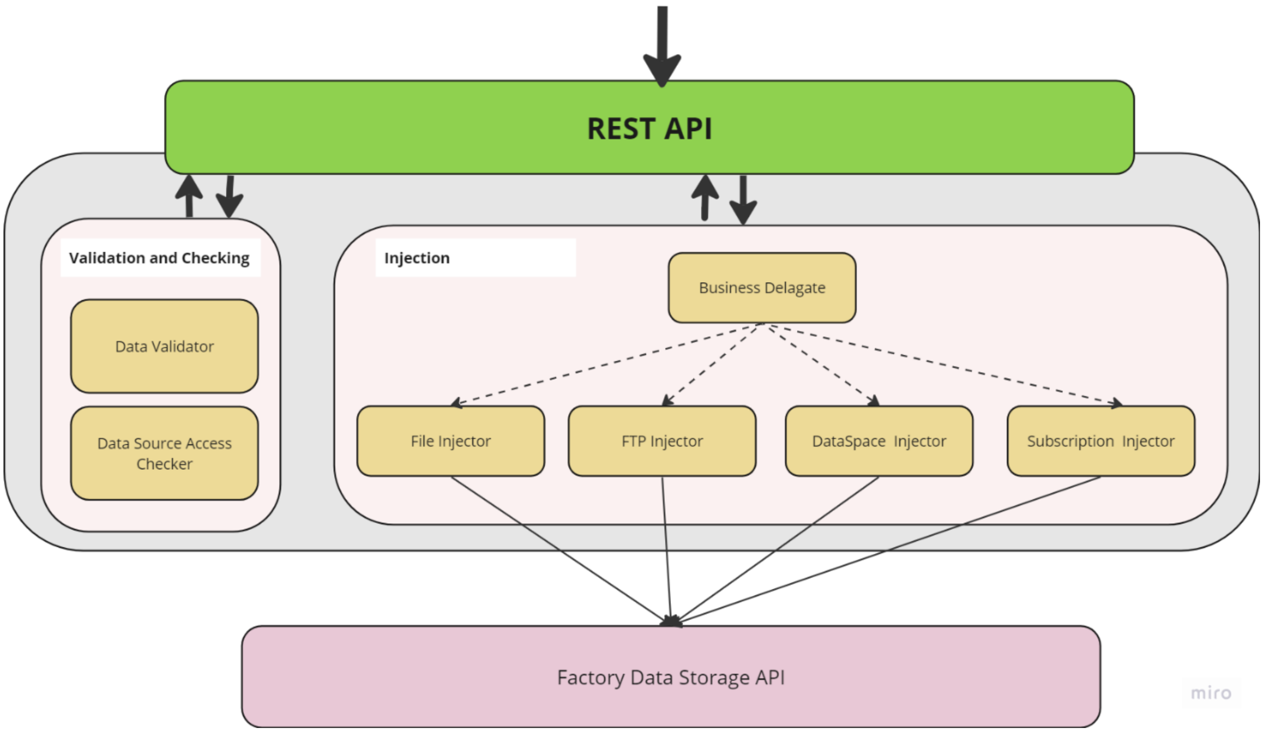


Figure 2. Data Check-In Architecture

Future iterations of Data Check-In will investigate the usage of Nifi [3] technology to support data injection through a queue-type system like Kafka or MQTT, an FTP server, or a subscription to a REST API. A distributed system called Apache NiFi is devoted to data extraction, transformation, and loading (ETL).

Ingestion, routing, and administration of data flows between various systems may be done effectively and visually with NiFi's design. It can be used to accomplish this by adding bespoke connectors in addition to the more than 300 external connectors that are already in place.

The ability to arrange data flows by dragging and connecting the required components onto the admin website canvases is one of NiFi's strong points. Therefore, understanding and properly configuring each component that you wish to use is more important than having specific programming knowledge.

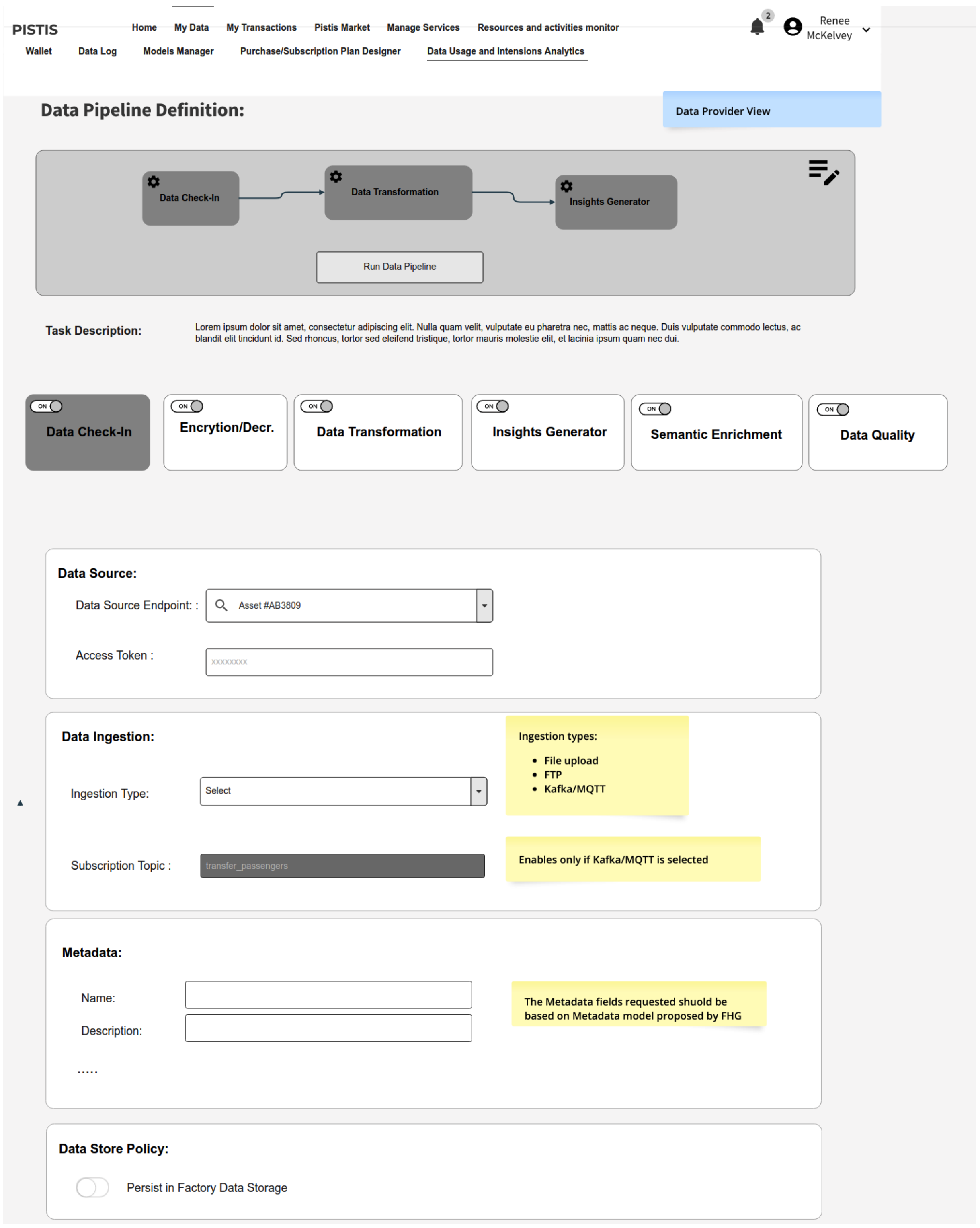
For instance, it is quite easy to use Apache Nifi to produce messages for Kafka and receive messages from Kafka; all you have to do is drag the processors into the UI and set their settings accordingly. The concept is the same for every processor; in this manner, you may assign all ETL tasks to Apache Nifi in order to maximise the utilisation of data sources and optimise the code in your applications.

[3] https://nifi.apache.org/

2.1.7 Mock-ups and Screenshots

The UI of this component will be a part of the data pipeline UI bundle, as seen in Figure 3. The objective is to be able to identify the component in charge of facilitating data ingestion as part of a data pipeline. The design of the corresponding user interface (UI) is built on four separate parts, which are applicable to both Data Check-In and any other potentially orchestrable component defined in a data pipeline using the Job Configurator:

1) Data Source: Common block for all selectable components in a data pipeline. The endpoint and its access credentials, among other pertinent details, must be supplied in this block in order to access the data source.
2) Data Ingestion: Specific block for the component. Depending on the type of ingestion chosen, the block known as "data injection" for data check-in will dynamically display the various fields that need to be filled out.
3) Metadata: A block for allowing the user to register specific metadata regarding the dataset that will be ingested during the chechin operation.
4) Storage Policy: Common block for all selectable components in a data pipeline. This block allows you to choose whether the output that is produced after a service is executed should be stored in the Factory Data Storage or (temporarily) in memory.



Data Pipeline Definition:

Task Description: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui.

Data Source:

Data Source Endpoint: : Asset #AB3809

Access Token : xxxxxxxx

Data Ingestion:

Ingestion Type: Select

Subscription Topic : transfer_passengers

Ingestion types:
- File upload
- FTP
- Kafka/MQTT

Enables only if Kafka/MQTT is selected

Metadata:

Name:

Description:

.....

The Metadata fields requested shuold be based on Metadata model proposed by FHG

Data Store Policy:

Persist in Factory Data Storage

Figure 3: Data Check-In UI Mock-up

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 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.

2.2.3 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	Beta	Data stored properly in Factory Data Storage after applying transfomations defined by the user.	Upcoming	PISTIS.OUS.03
UC_04	Data consumer	Get an improved set of transformations available	I can aply a wider set of transformations to my dataset	Beta	The catalogue of available transformations in beta	Upcoming	PISTIS.OUS.03

					version is bigger than the catalogue in alpha version		

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's Main Elements and Internal 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 catalog offered via GET API call.

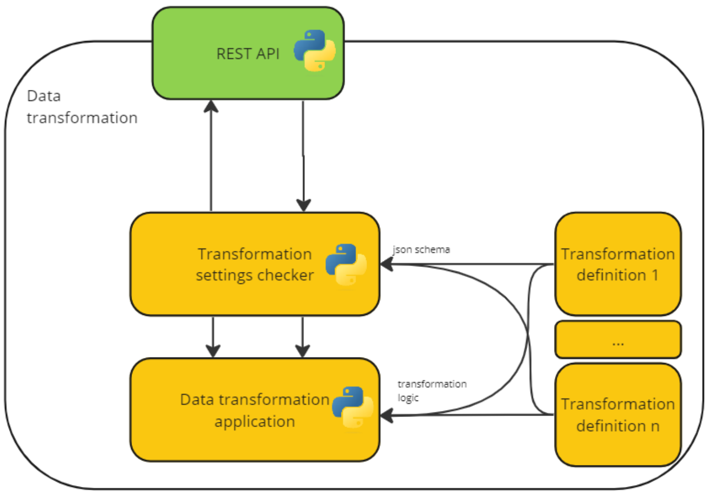
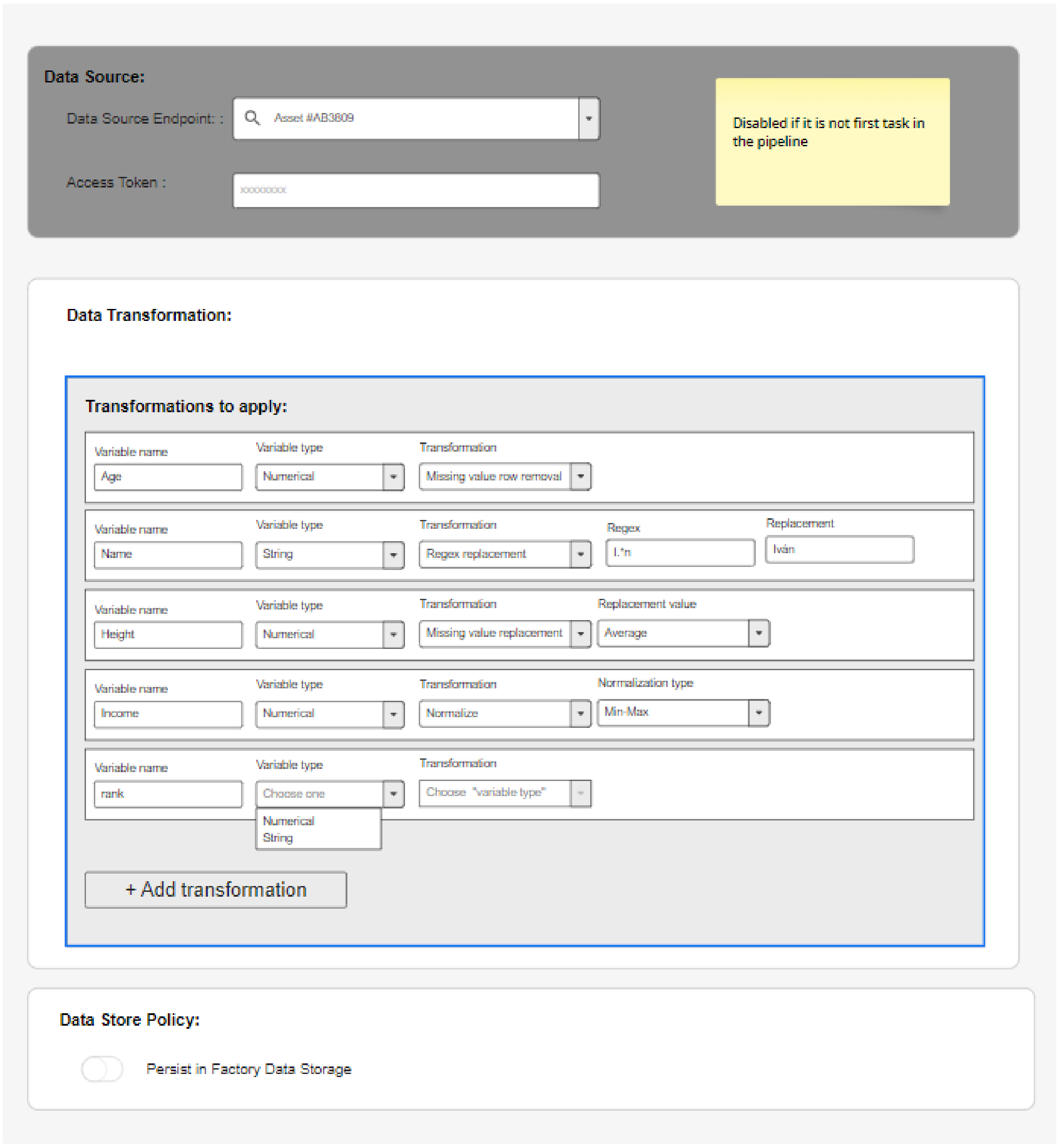

REST API
Data transformation
Transformation settings checker
json schema
Transformation definition 1
...
Data transformation application
transformation logic
Transformation definition n

Figure 4: Data Transformation Component's Internal Architecture

2.2.7 Mock-ups and Screenshots

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 case that is not possible, the GUI should allow the input of the JSON of the different transformations to apply to the dataset to be processed.



Data Source:

Data Source Endpoint: :	Asset #AB3809
Access Token :	xxxxxxxx

Disabled if it is not first task in the pipeline

Data Transformation:

Transformations to apply:

Variable name	Variable type	Transformation		
Age	Numerical	Missing value row removal		
Name	String	Regex replacement	Regex: I.*n	Replacement: Iván
Height	Numerical	Missing value replacement	Replacement value: Average	
Income	Numerical	Normalize	Normalization type: Min-Max	
rank	Choose one (Numerical, String)	Choose "variable type"		

Data Store Policy:

Persist in Factory Data Storage

Figure 5: Data Transformation Component UI Mock-up

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 Technology Background

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

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.	Upcoming	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.

2.3.6 Component's Main Elements and Internal Architecture

Figure 6 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).

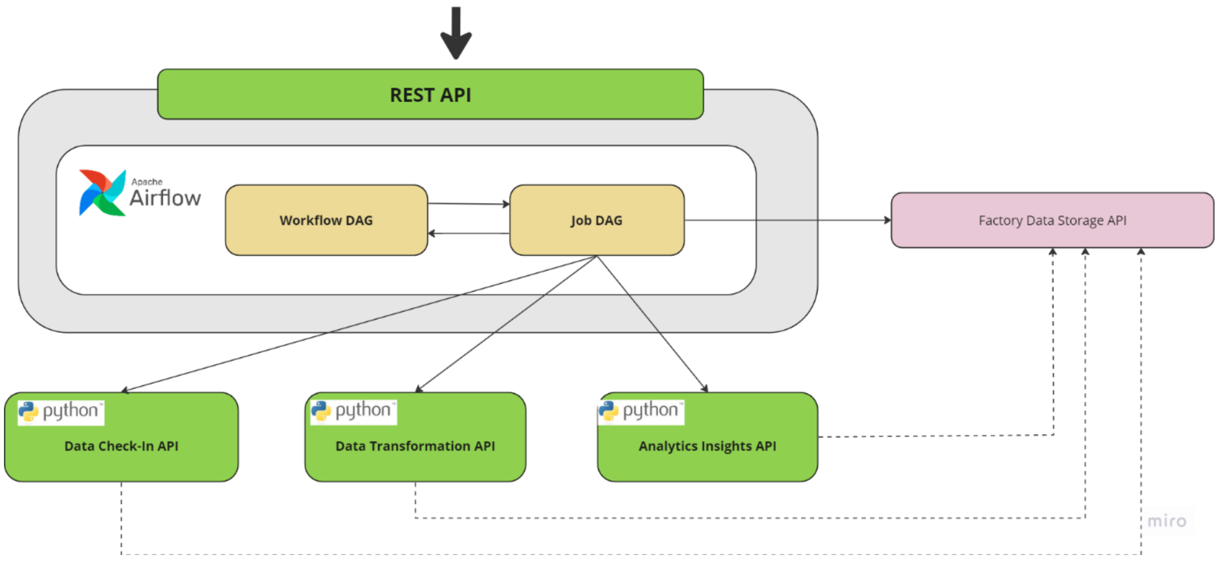


Figure 6: Job Configurator Architecture

The workflow execution is carried out via the WDAG, whose internal structure is depicted in Figure 7. Like the WDAG, the JDAG oversees carrying out a job or component in the context of PISTIS. Figure 7 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.

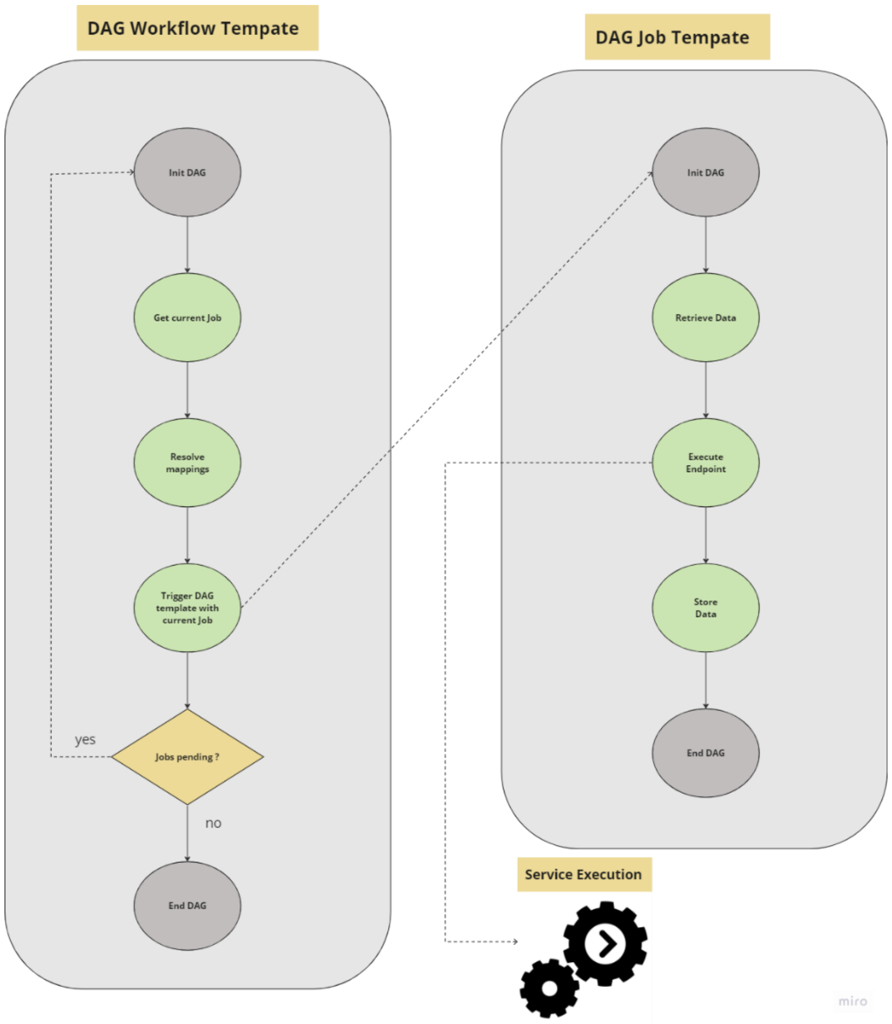


Figure 7: Workflow and Job DAGs

Finally, it is important to highlight that any service potencially orchestrable using the Job Configurator should satisfy the design principles shown in Figure 8.

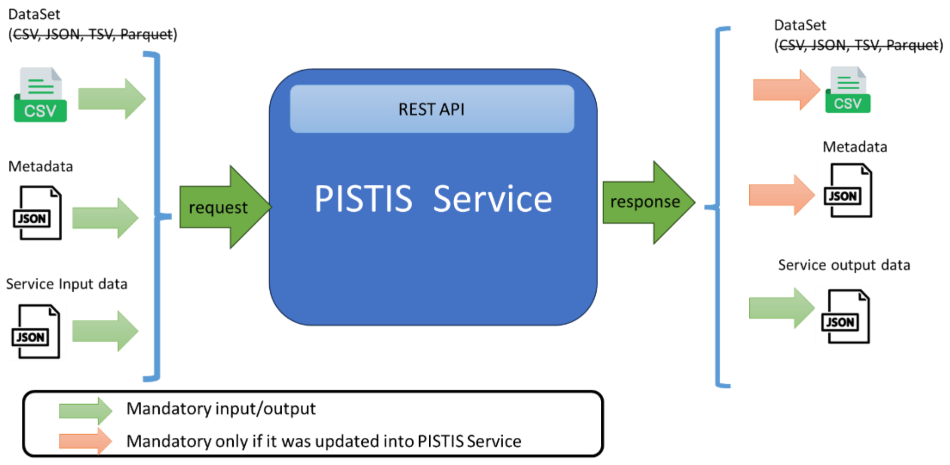


Figure 8: Service design principles approach 1

The choosen by the consortium format for working with data sources at the workflow level in the Alpha version of the component is CSV, which was chosen over JSON, TSV, and Parquet. CSV is the most simple and common format used by demonstrator partners in PISTIS.

Figure 9 depicts an alternative way to defining an orchestrable service in the context of PISTIS, with the main concept being format conversion.

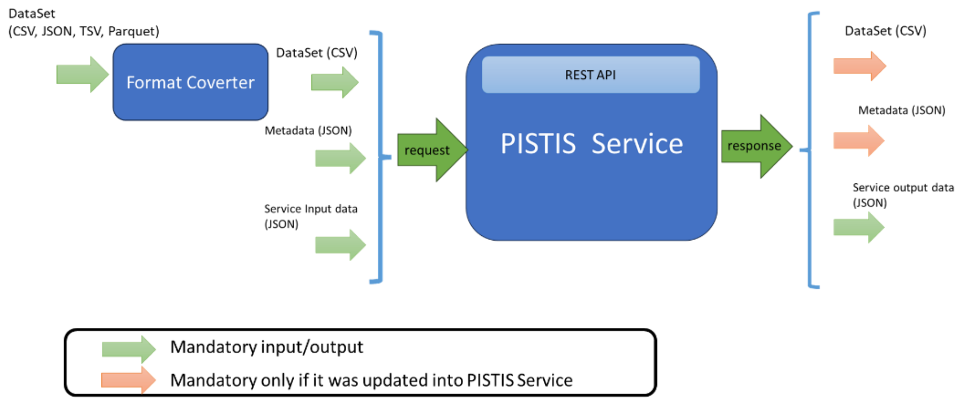


Figure 9: Service design principles approach

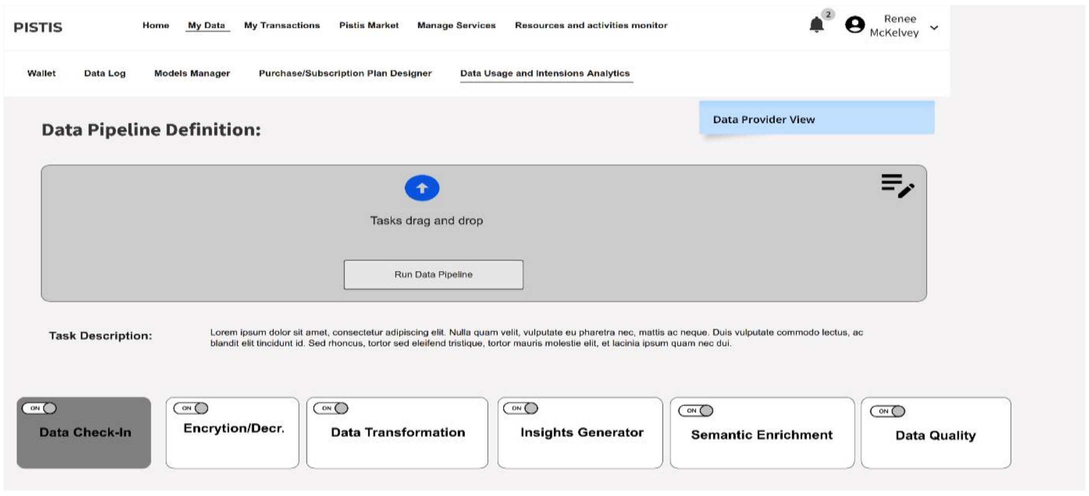
For the Alpha version, the Job Configurator will be based on the selection and execution of services that support Approach 1, with CSV as the fixed format for data sources.

2.3.7 Mock-ups and Screenshots

The UI definition for the Job Configurator has been included within the Data Pipeline UI bundle. The Job Configurator UI will be built on the idea of easily creating a workflow of jobs by dragging and dropping services and linking them to the next one with an arrow. Initially, the user begins with an empty panel onto which the user drags a series of services displayed just below the panel, as illustrated in Figure 10.

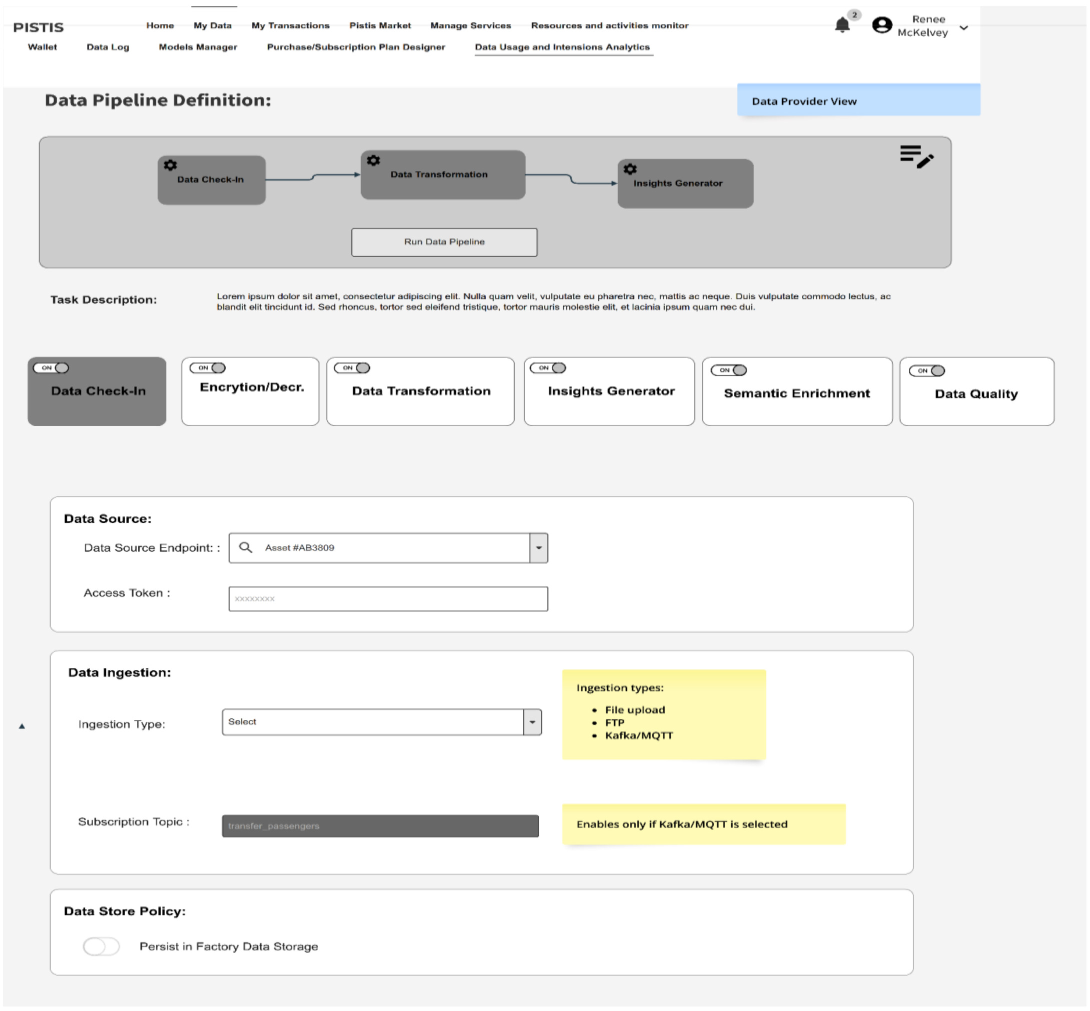
When a service is moved to the panel, either by selecting it or situating itself on it, a new panel at the bottom appears dynamically with the appropriate fields whose contents must be supplied for the service to run correctly. Figure 11 is an example of this using the data check-in service.

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



Task Description: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui.

Figure 10: Job Configurator GUI Mock-up



Task Description: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui.

Data Source:

Data Source Endpoint: : Asset #AB3809

Access Token : xxxxxxxx

Data Ingestion:

Ingestion Type: Select

Ingestion types:
- File upload
- FTP
- Kafka/MQTT

Subscription Topic : transfer_passengers

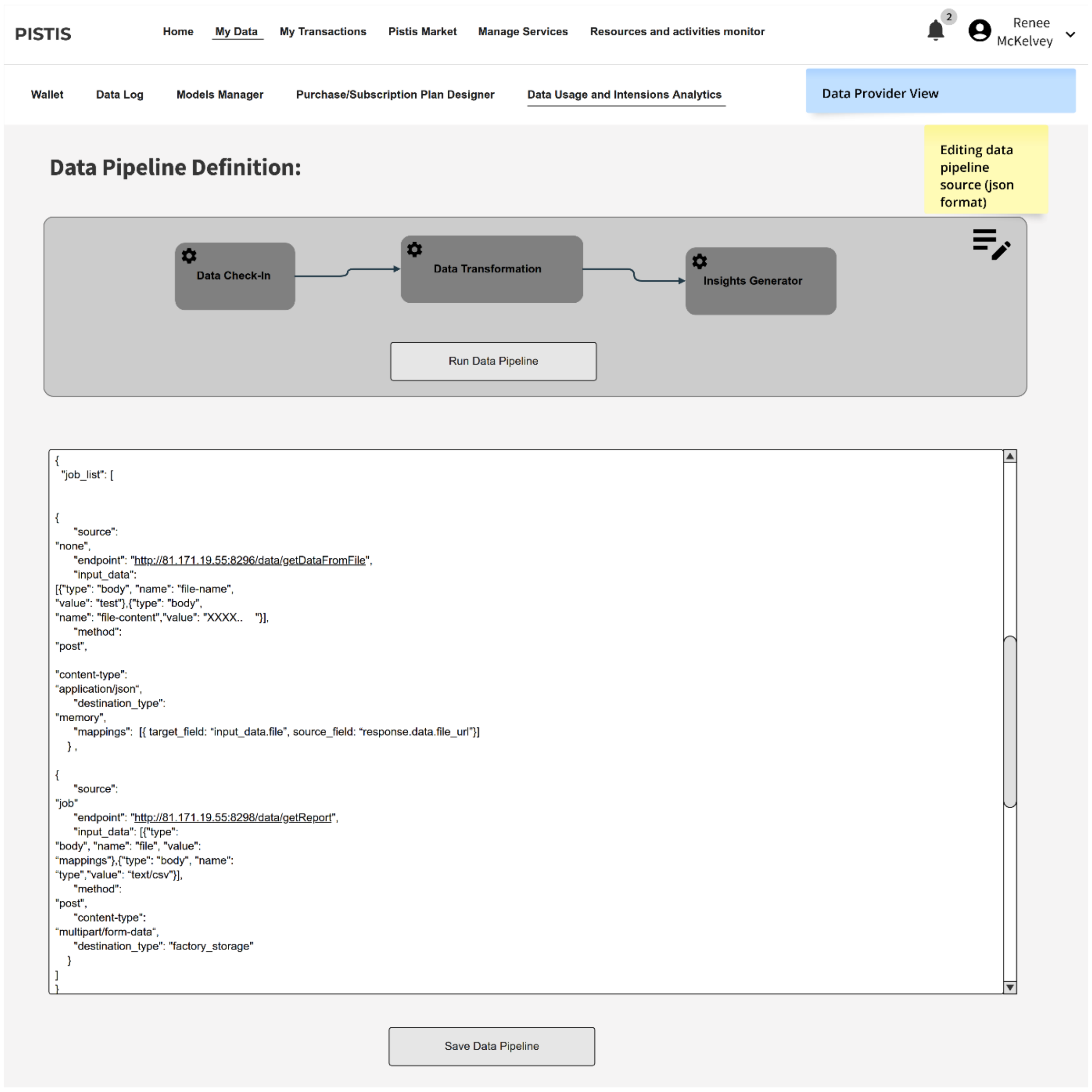
Enables only if Kafka/MQTT is selected

Data Store Policy:

Persist in Factory Data Storage

Figure 11: Data Check-In GUI Mock-up

Finally, as shown in Figure 12, an option represented by a pencil has been added to the upper right corner of the main panel to allow revision of the source code on which the specification of the workflow or data pipeline is based, which is written in Json.



Data Pipeline Definition:

```
{
  "job_list": [


{
      "source":
"none",
      "endpoint": "http://81.171.19.55:8296/data/getDataFromFile",
      "input_data":
[{"type": "body", "name": "file-name",
"value": "test"},{"type": "body",
"name": "file-content","value": "XXXX..    "}],
      "method":
"post",

"content-type":
“application/json“,
      "destination_type":
"memory",
      "mappings":  [{ target_field: “input_data.file”, source_field: “response.data.file_url”}]
    } ,

{
      "source":
"job"
      "endpoint": "http://81.171.19.55:8298/data/getReport",
      "input_data": [{"type":
"body", "name": "file", "value":
“mappings"},{"type": "body", "name":
“type","value": “text/csv"}],
      "method":
"post",
      "content-type":
“multipart/form-data“,
      "destination_type": "factory_storage"
    }
]
}
```

Figure 12: Data Pipeline Editing GUI Mock-up

2.4 Analytics Engine

2.4.1 Component Description

In order to run ML/DL analytics pipelines and transform the primary data artifacts 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 TECHNOLOGY BACKGROUND

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

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	Upcoming	PISTIS.OUS.03
UC_03	Data Consumer	Train an analytics model	Create a new AI Model	Beta	Check AI Model	Upcoming	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 analytics results	Upcoming	PISTIS.OUS.03

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's Main Elements and Internal 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 Artifacts 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)

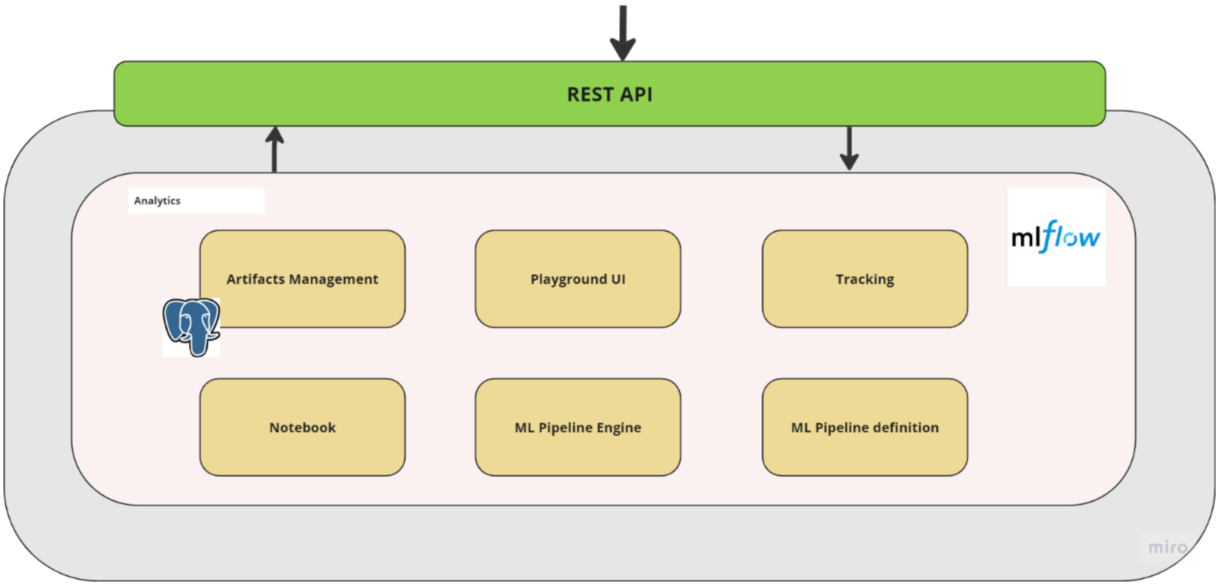
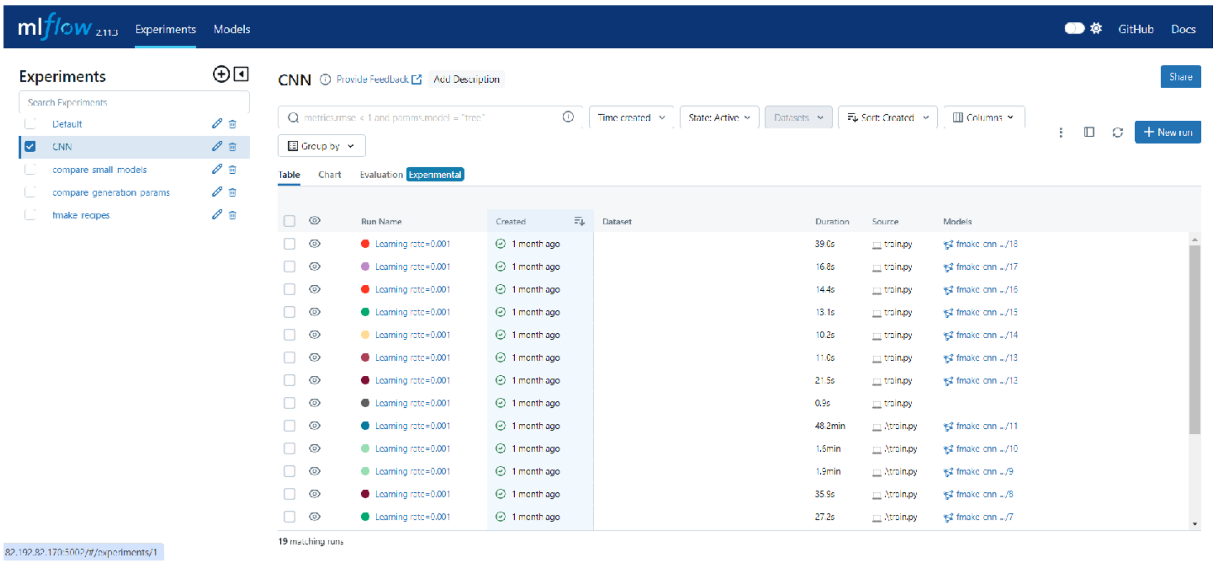


Figure 13: Analytics Engine Architecture

2.4.7 Mock-ups and Screenshots

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 artifacts 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 artifacts that lead to that output.

Figure 14 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 14: Analytics Engine GUI.

2.5 DATA ENRICHMENT

2.5.1 COMPONENT DESCRIPTION

The data enrichment component in PISTIS is included in the Data Ingestion and Transformation module, which is in the premises of an organization. Along with the data transformation component, this component is responsible to transform and enrich any available data assets to increase its value for trading. Data enrichment refines and enhances datasets to add more value and utilization to the existing data. Typically, data enrichment refers to data harmonization using additional data sources. It combines information from several data sources into a standardized format for further data analysis. The different source of data could be in different file formats, naming conventions and disparate data sources.

The main objective of the Data Enrichment module in PISTIS is to parse a dataset available as a file into a structured table with the respective domain specific data models. This allows the users the flexibility to upload datasets as files and later transform them into PISTIS datasets. This module supports the transformation of CSV, TXT and XLS files but for the alpha version, only CSV files are available. This component constitutes of a frontend and a backend, the backend serves the functionalities to parse a file into a table, along with keeping an updated database of the PISTIS data models and the frontend provides the interface for the user to select an appropriate data model for a table. After a file is transferred into a table with the correct data model, the table is inserted into the Factory Data Storage. This way, the user will always have access to the initial dataset as a file and the transformed table with a standardized data model. It also allows the user the flexibility to transfer a file into multiple tables with different data models.

2.5.2 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.

2.5.3 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.0 1
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.0 6
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.0 6
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.0 1

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's Main Elements and Internal Architecture

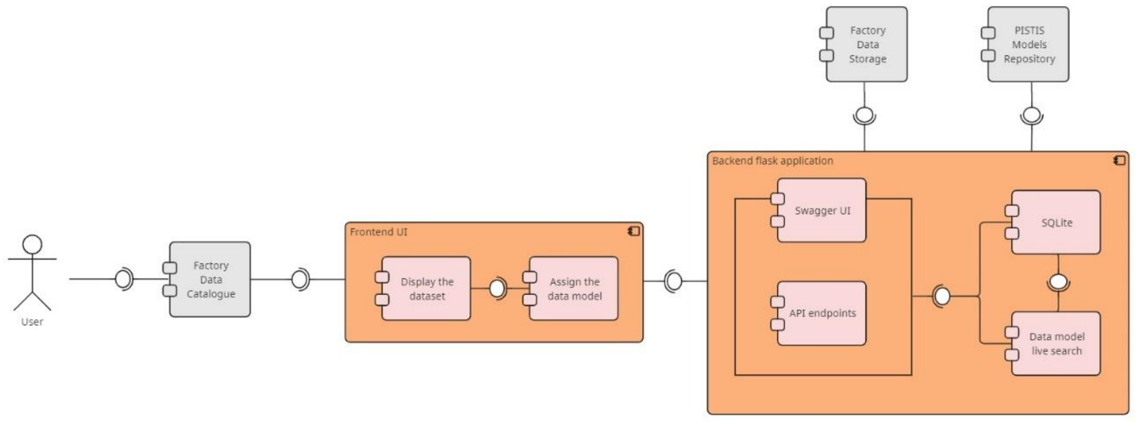
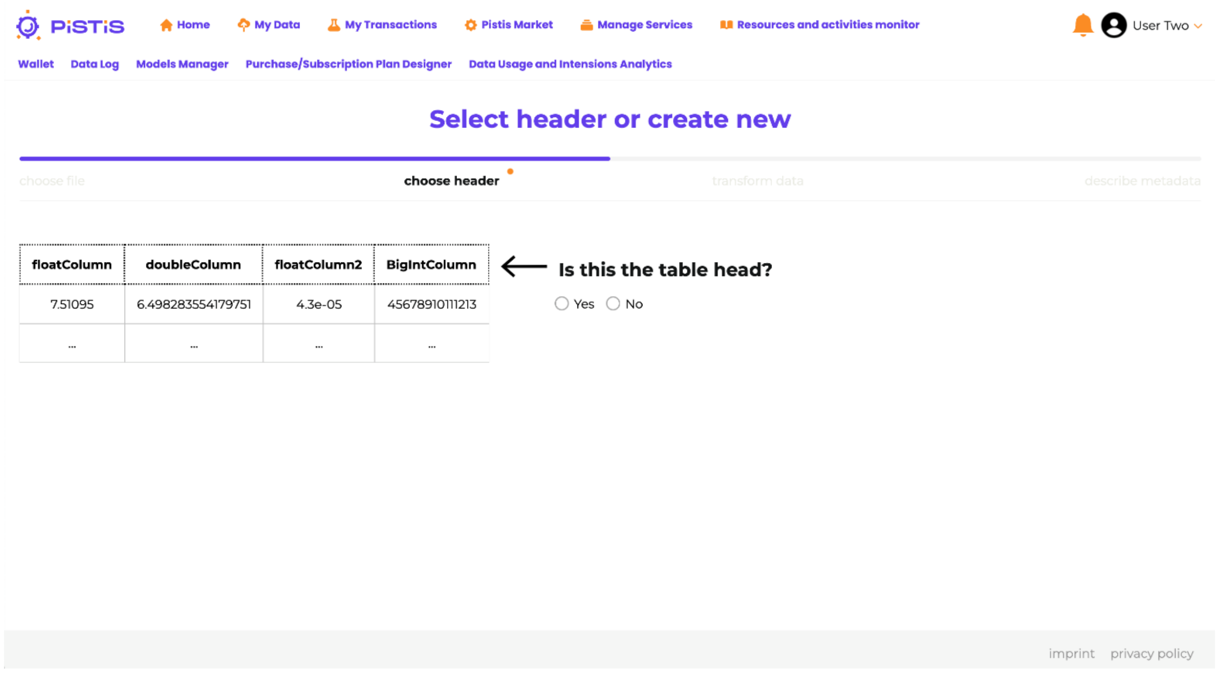


Figure 15: Data Enrichment Internal Architecture

2.5.7 Mock-ups and Screenshots



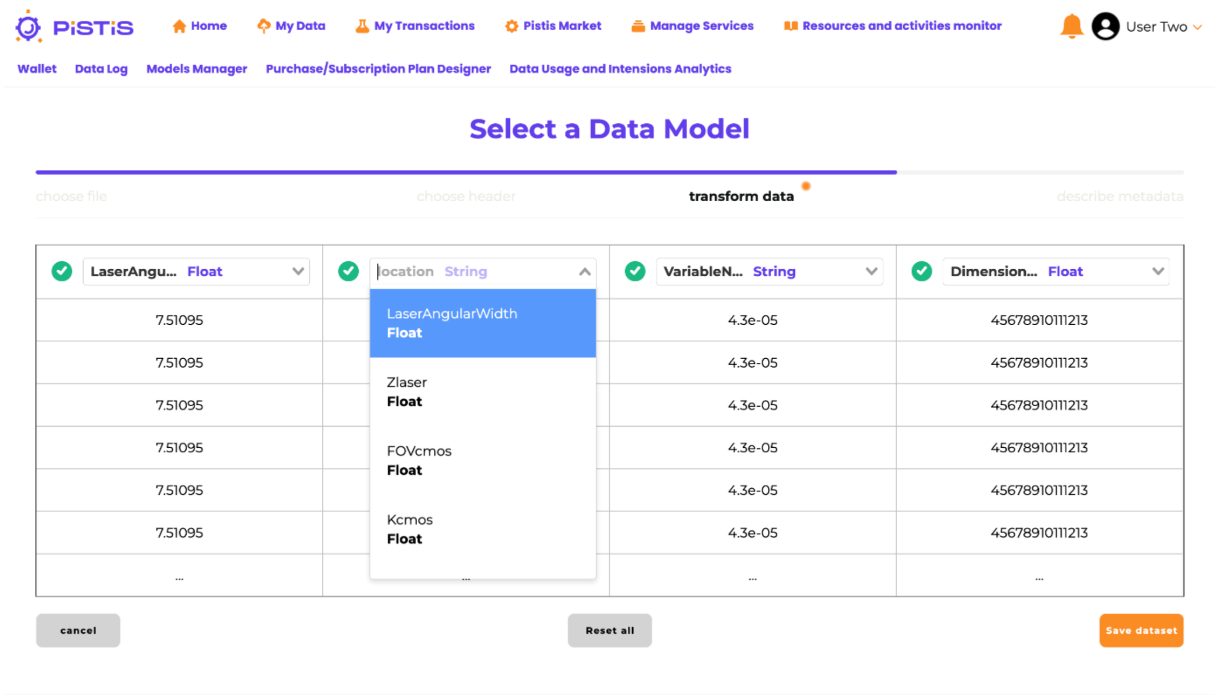
Select header or create new

floatColumn	doubleColumn	floatColumn2	BigIntColumn
7.51095	6.498283554179751	4.3e-05	45678910111213
...	...	...	...

Is this the table head?

Yes No

Figure 16: Display the dataset



Select a Data Model

LaserAngu... Float	location String	VariableN... String	Dimension... Float
7.51095	LaserAngularWidth Float	4.3e-05	45678910111213
7.51095		4.3e-05	45678910111213
7.51095	Zlaser Float	4.3e-05	45678910111213
7.51095	FOVcmos Float	4.3e-05	45678910111213
7.51095		4.3e-05	45678910111213
7.51095	Kcmos Float	4.3e-05	45678910111213
...	...	...	...

Figure 17: Select the properties of the data model

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 information provided in the metadata and returns the validation result together with a score of the result. 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. User-defined validation rules can be incorporated to address specific data quality requirements.

2.6.2 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 catalog is used to store the result as linked data, a MongoDB is used for aggregating and caching the result to display it.
- The data quality 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.

2.6.3 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	In Progre ss	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.04
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	In progress	PISTIS.OUS.04
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.04
UC_05	Data Provider	Represent the best possible quality of my data	I can archieve a higher valuation	Beta	Display QA report for data	Upcoming	PISTIS.OUS.04
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	Upcoming	PISTIS.OUS.04

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 non-functional requirements to the component are not identified yet.

2.6.6 Component's Main Elements and Internal Architecture

Brief description of internal component's elements and the architecture of the component

2.6.6.1 *Metadata Quality Assessment*

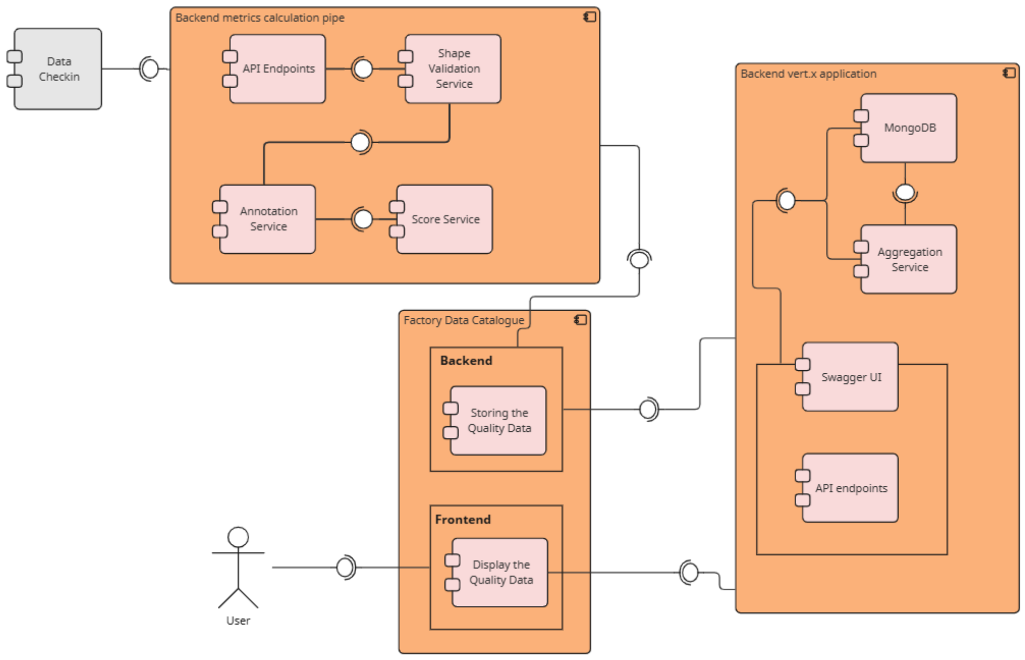


Figure 18: Metadata Quality Assessment service internal architecture

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.

2.6.6.2 Data Quality assessment

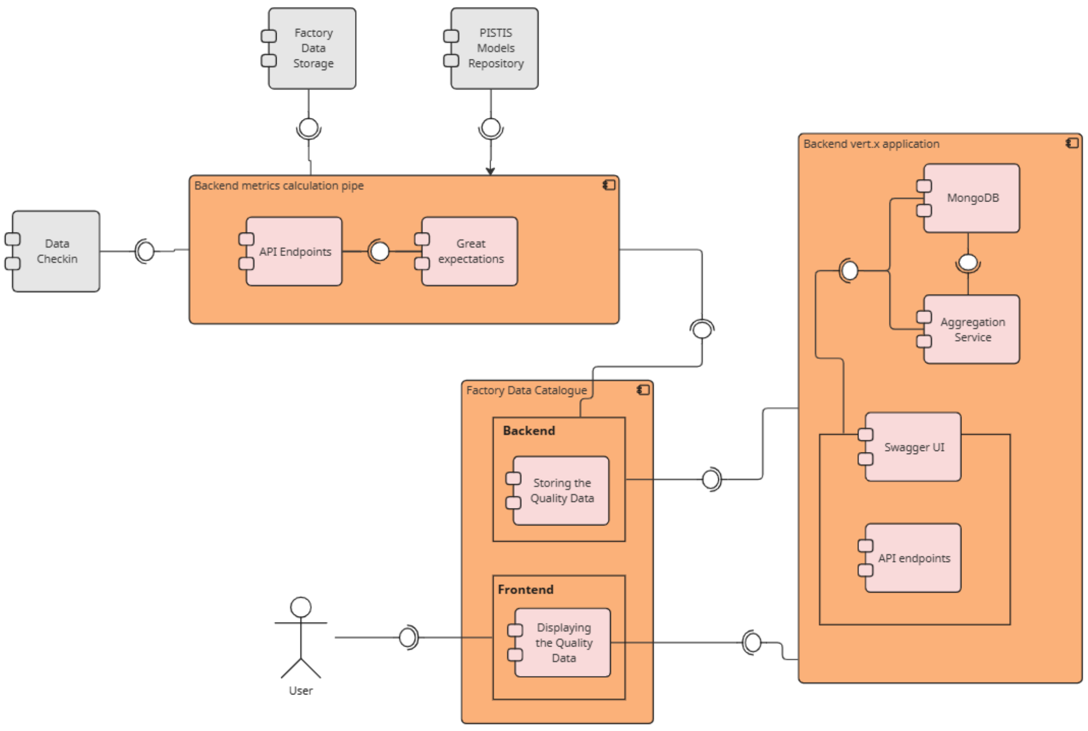
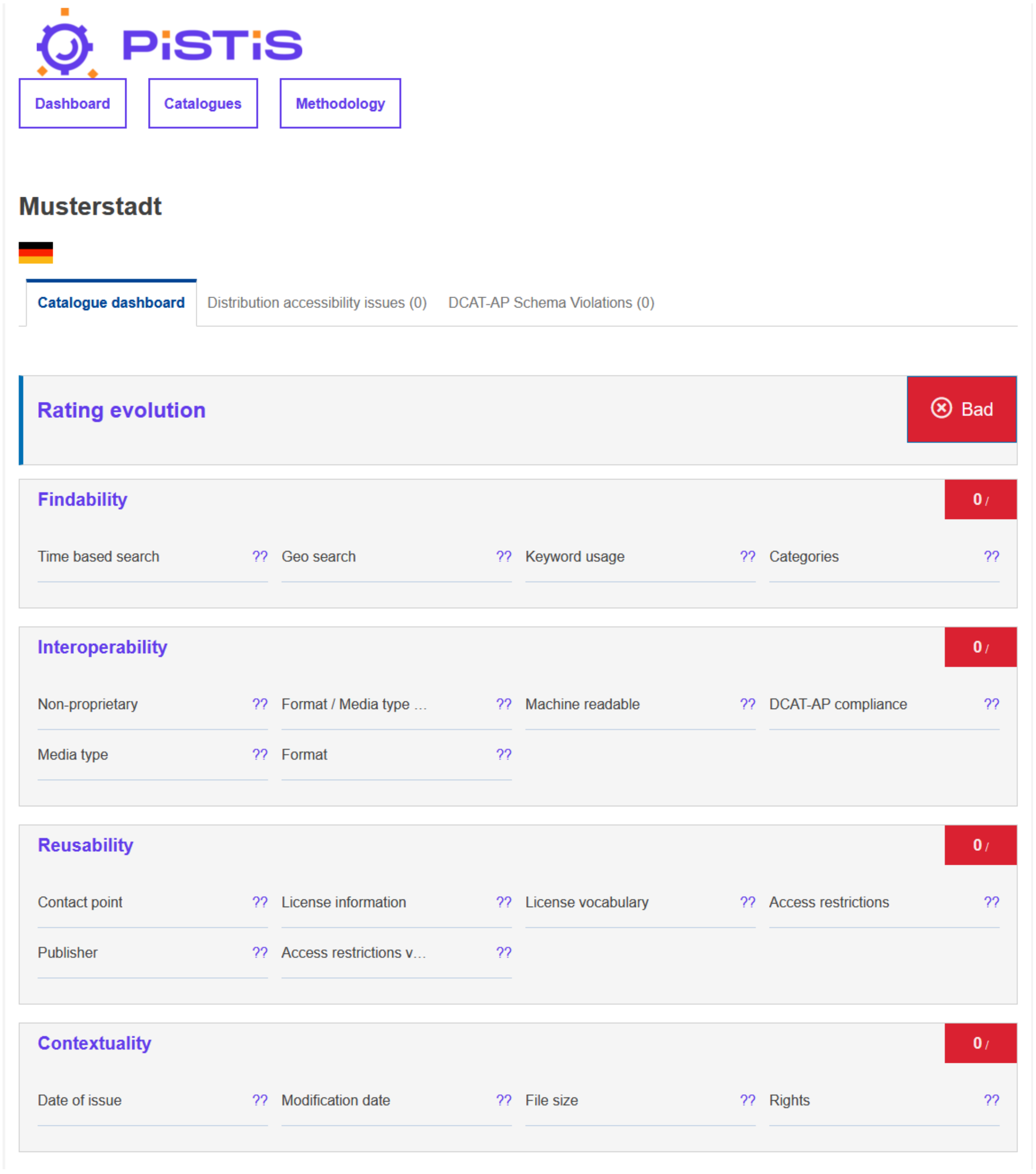


Figure 19: Data Quality Assessment service internal architecture

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.

2.6.7 Mock-ups and Screenshots



Musterstadt

Catalogue dashboard | Distribution accessibility issues (0) | DCAT-AP Schema Violations (0)

Rating evolution — Bad

Findability 0 /

Time based search	??	Geo search	??	Keyword usage	??	Categories	??

Interoperability 0 /

Non-proprietary	??	Format / Media type ...	??	Machine readable	??	DCAT-AP compliance	??
Media type	??	Format	??				

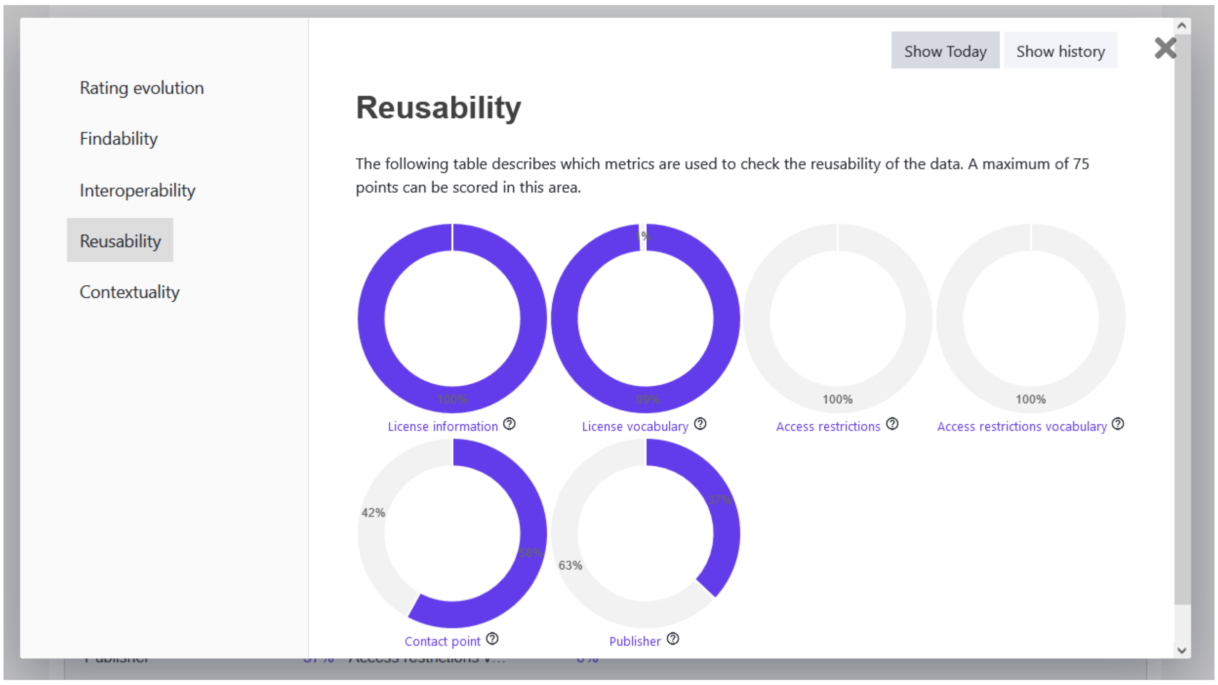
Reusability 0 /

Contact point	??	License information	??	License vocabulary	??	Access restrictions	??
Publisher	??	Access restrictions v...	??				

Contextuality 0 /

Date of issue	??	Modification date	??	File size	??	Rights	??

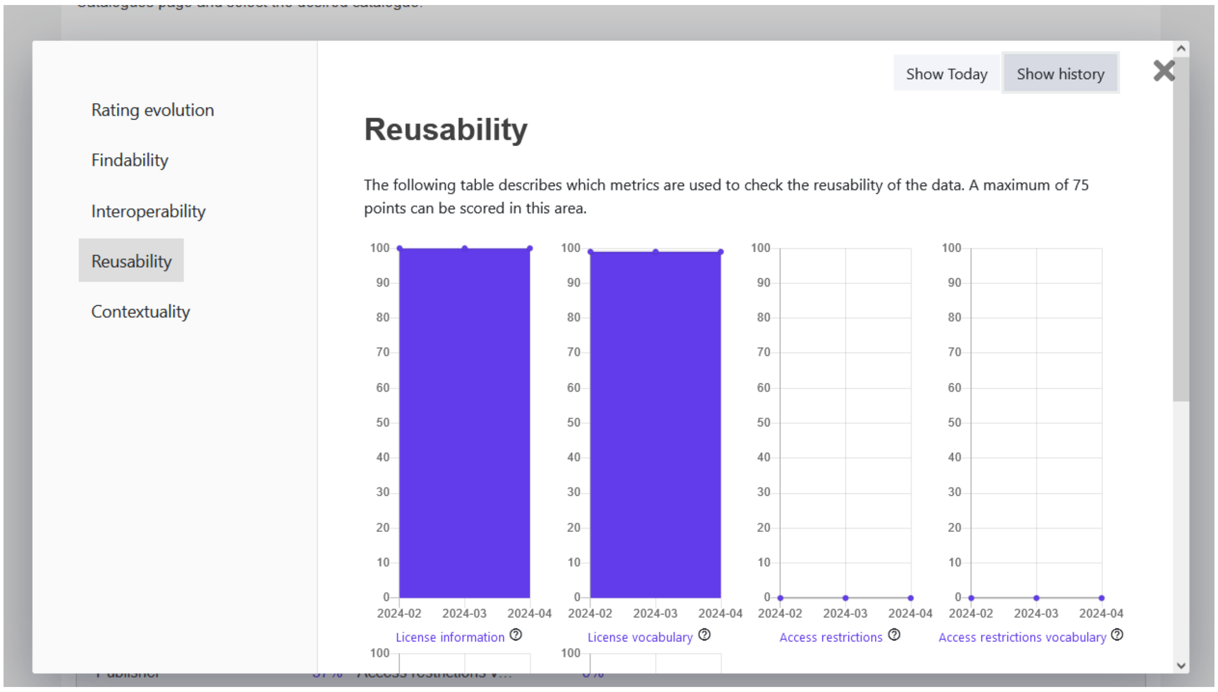
Figure 20: Overview of the assessment result with the score "bad"



Reusability

The following table describes which metrics are used to check the reusability of the data. A maximum of 75 points can be scored in this area.

Figure 21: Dive in into the result for the metrics of the dimension reusability



Reusability

The following table describes which metrics are used to check the reusability of the data. A maximum of 75 points can be scored in this area.

Figure 22: Historic results for the dimension reusability

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 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.

2.7.3 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	Upcoming	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	1.0	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's Main Elements and Internal Architecture

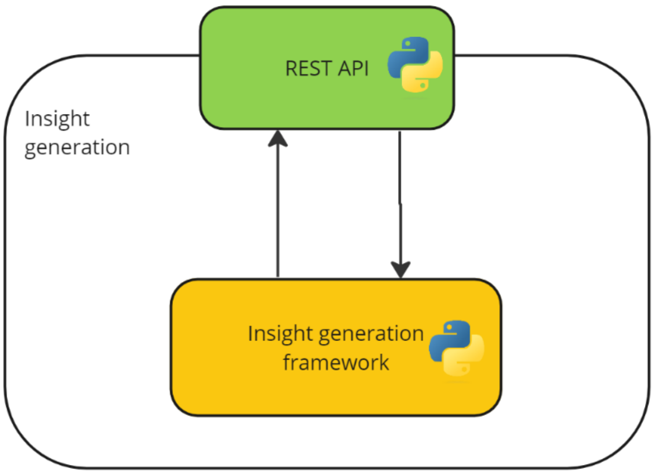


Figure 23: Component's Internal Architecture

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 Mock-ups and Screenshots

Due to the nature of the Insight Generator component it doesn't have a GUI.

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** runs within the premises of the data provider and serves as the access point to the organisation's data assets. It provides the means to make the metadata and data available. Each organisation is responsible for maintaining its own catalogue and incorporating their own (meta)data.

The **PISTIS Data Catalogue** is a centralized service within the PISTIS Cloud Platform and aggregates the metadata from all Factory Data Catalogues. It constitutes the central marketplace of PISTIS by allowing to browse all available data assets offered for sharing by the data providers. The visibility of the data assets is determined by the respective data access policies.

The Factory Data Catalogues and the PISTIS Data Catalogue will be connected via the Asset Description Bundler, that is responsible for making data from the factories available for acquisiton in the PISTIS Cloud Platform, through publishing the relevant metadata only. As such, the data remain with the data providers until a transaction is performed

3.1.2 TECHNOLOGY BACKGROUND

The Data Catalogue will be based on the scalable, Open Source and Java-based metadata management solution piveau[4]. Piveau is catalogue solution, designed around Semantic Web technologies and applies a Triplestore as its primary database to leverage the full potential of

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

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.

On top a high-performance search service is integrated based on Elasticsearch, allowing to perform search and filter operations on the data. Furthermore, piveau comes with a user-friendly user interface for browsing and creating metadata, which is developed with Vue.js.

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.

3.1.3 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	In Progress	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	Upcoming	PISTIS.OUS.10

					data is presented		

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	
FR_03	Harvest/import existing metadata from another source	US_01	

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's Main Elements and Internal 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 24, with its internal components is detailed further in the Factory Data Catalogue component section.

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 Exevution Engine).

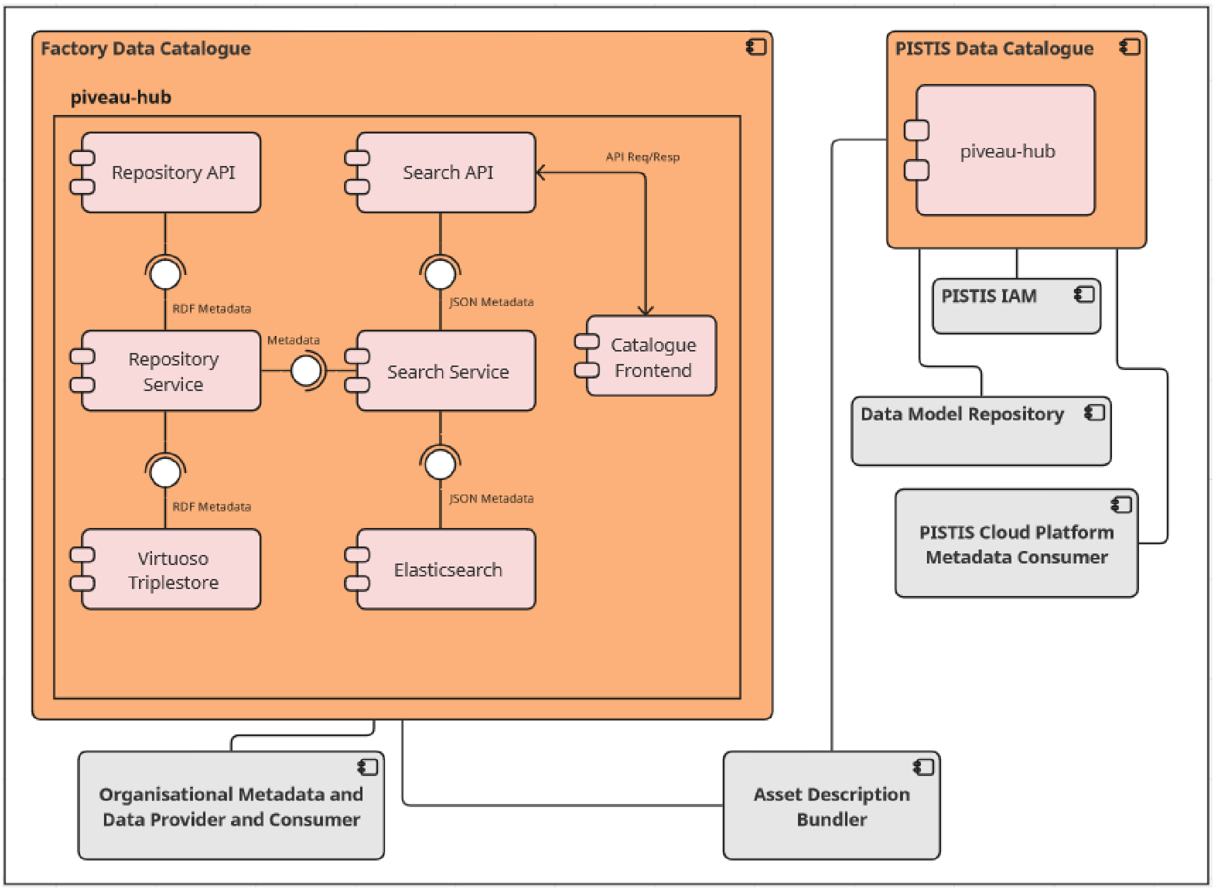
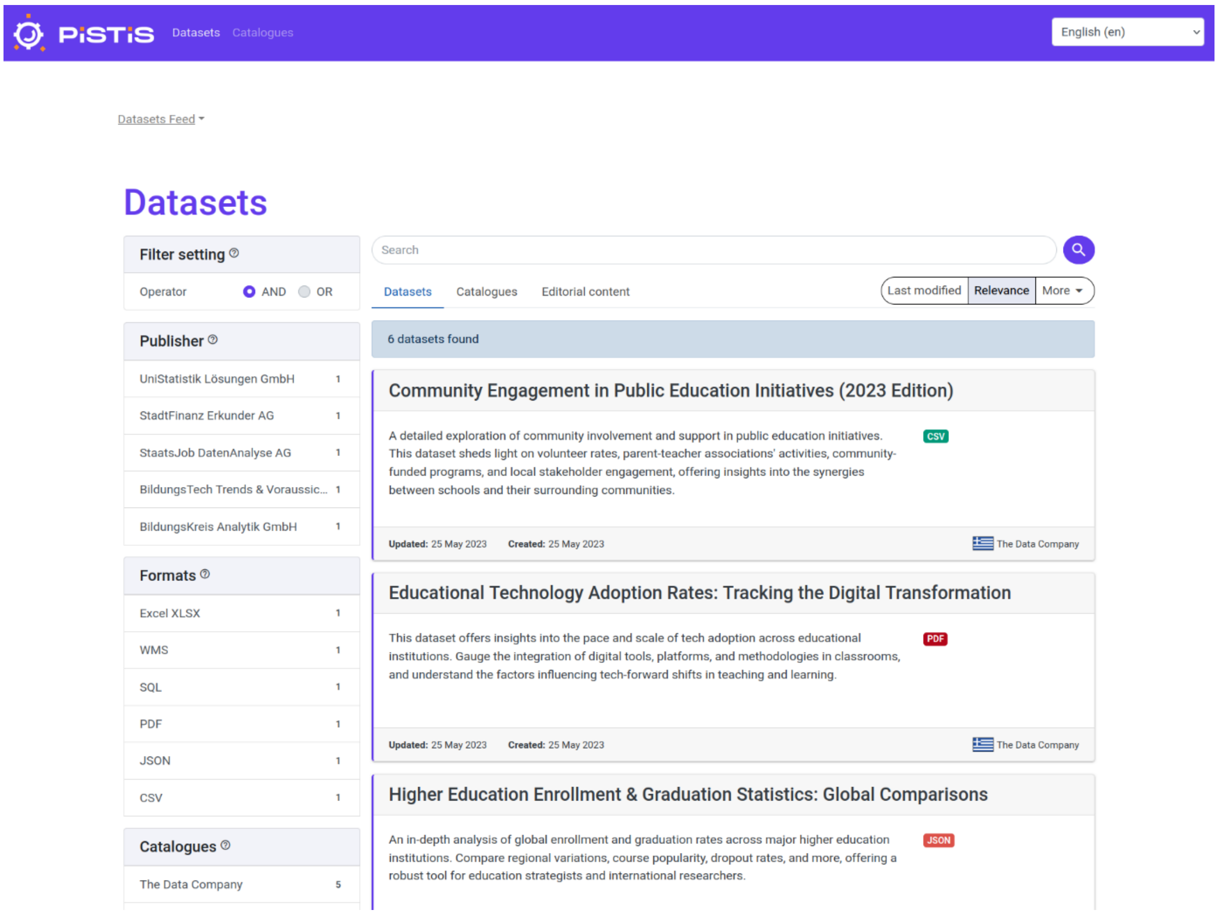


Figure 24: Data Catalogue's Internal Architecture

3.1.7 Mock-ups and Screenshots



Datasets

Filter setting

Operator [✓ AND] OR

Publisher

UniStatistik Lösungen GmbH	1
StadtFinanz Erkunder AG	1
StaatsJob DatenAnalyse AG	1
BildungsTech Trends & Voraussic...	1
BildungsKreis Analytik GmbH	1

Formats

Excel XLSX	1
WMS	1
SQL	1
PDF	1
JSON	1
CSV	1

Catalogues

The Data Company	5

Search

Datasets | Catalogues | Editorial content — Last modified | Relevance | More

6 datasets found

Community Engagement in Public Education Initiatives (2023 Edition)

A detailed exploration of community involvement and support in public education initiatives. This dataset sheds light on volunteer rates, parent-teacher associations' activities, community-funded programs, and local stakeholder engagement, offering insights into the synergies between schools and their surrounding communities. CSV

Updated: 25 May 2023 **Created:** 25 May 2023 The Data Company

Educational Technology Adoption Rates: Tracking the Digital Transformation

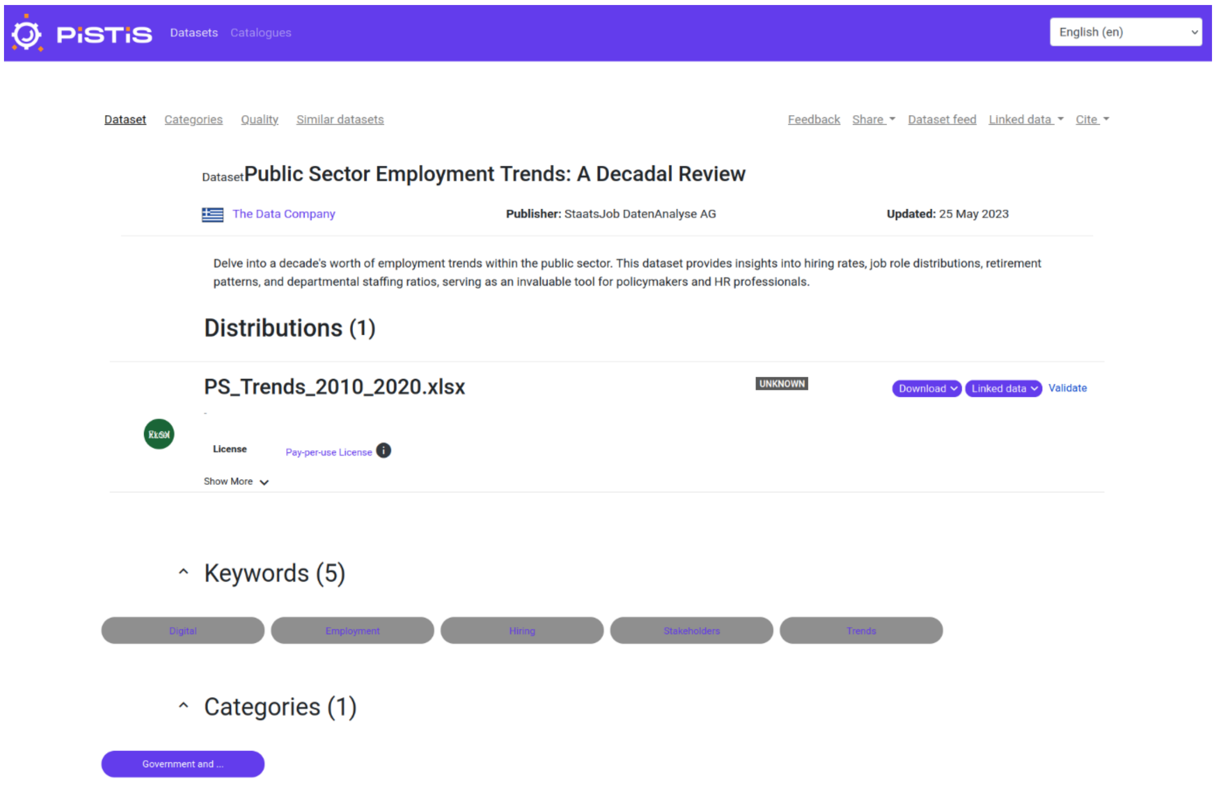
This dataset offers insights into the pace and scale of tech adoption across educational institutions. Gauge the integration of digital tools, platforms, and methodologies in classrooms, and understand the factors influencing tech-forward shifts in teaching and learning. PDF

Updated: 25 May 2023 **Created:** 25 May 2023 The Data Company

Higher Education Enrollment & Graduation Statistics: Global Comparisons

An in-depth analysis of global enrollment and graduation rates across major higher education institutions. Compare regional variations, course popularity, dropout rates, and more, offering a robust tool for education strategists and international researchers. JSON

Figure 25: A list of datasets with filters on the left-side


PiSTiS Datasets Catalogues English (en)

Dataset Categories Quality Similar datasets Feedback Share Dataset feed Linked data Cite

Dataset Public Sector Employment Trends: A Decadal Review

The Data Company **Publisher:** StaatsJob DatenAnalyse AG **Updated:** 25 May 2023

Delve into a decade's worth of employment trends within the public sector. This dataset provides insights into hiring rates, job role distributions, retirement patterns, and departmental staffing ratios, serving as an invaluable tool for policymakers and HR professionals.

Distributions (1)

PS_Trends_2010_2020.xlsx UNKNOWN Download Linked data Validate

License Pay-per-use License

Show More

Keywords (5)

Digital Employment Hiring Stakeholders Trends

Categories (1)

Government and ...

Figure 26: Presentation of a dataset and its associated metadata

3.2 Factory Data Storage

3.2.1 Component Description

The Factory Data Storage is a component that provides a comprehensive solution with two relational databases integrated with a RESTful API to provide secure and efficient access to the data. It is hosted locally by the Data Factory environment which is on the premises of an organization that is a member of the PISTIS ecosystem. Data from each organization is ingested into the Data Storage by the Data Check-In module. This component comprises of a primary database that stores datasets in the form of tables and an additional database to store datasets in the form of files. Access to these databases is provided through a RESTful API.

The datasets are stored in different ways based on its format. If a dataset is in the form of a relational table, with a data schema, it is stored as a SQL table in the primary database. Each table is assigned a unique identifier in the form of a UUID which can be later used to query the rows and columns of this table. If a dataset is available in the form of a file (for example .csv, .txt, .xml), they are stored in the secondary database inside a table, also using a UUID. These files can be later converted into a SQL table and stored in the primary database.

The main functionalities of the Factory Data Storage are:

1. Storage for tables and files in a relational database
2. Access to the database using a REST API
3. CRUD operations on tables

4. CRUD operations on files

3.2.2 Technology Background

Factory Data Storage comprises of two major modules, the databases to provide storage to the files and a REST API to provide access to the databases. The database is built using PostgreSQL which is an open-source object relational database management system, and the RESTful API is built using Python Flask and SQL Alchemy which is a Python SQL toolkit and Object Relational Mapper (ORM) that provides the full power and flexibility of SQL. The two databases are hosted in an instance of the PostgreSQL and seamlessly integrated and provided access to using the RESTful API.

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 and quality analysis	Alpha	Has access rights to the table	Done	PISTIS.OUS.04

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's Main Elements and Internal Architecture

The Factory Data Storage is built using a Python Flask app, SQLAlchemy, and PostgreSQL to meet PISTIS's factory storage functionalities. 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 datamodel, data and any metadata such as the name of the table.

Inputs provided to every API 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 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, 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 for tables or files. The database for tables holds SQL tables that may vary in data schema and data and the database for files have tables with fixed schema that stores the file, the name of the file and the UUID of the file. Every dataset stored in the database has a UUID assigned to it, this UUID is the unique identifier of the table and is used to perform GET, UPDATE and DELETE operations on this dataset.

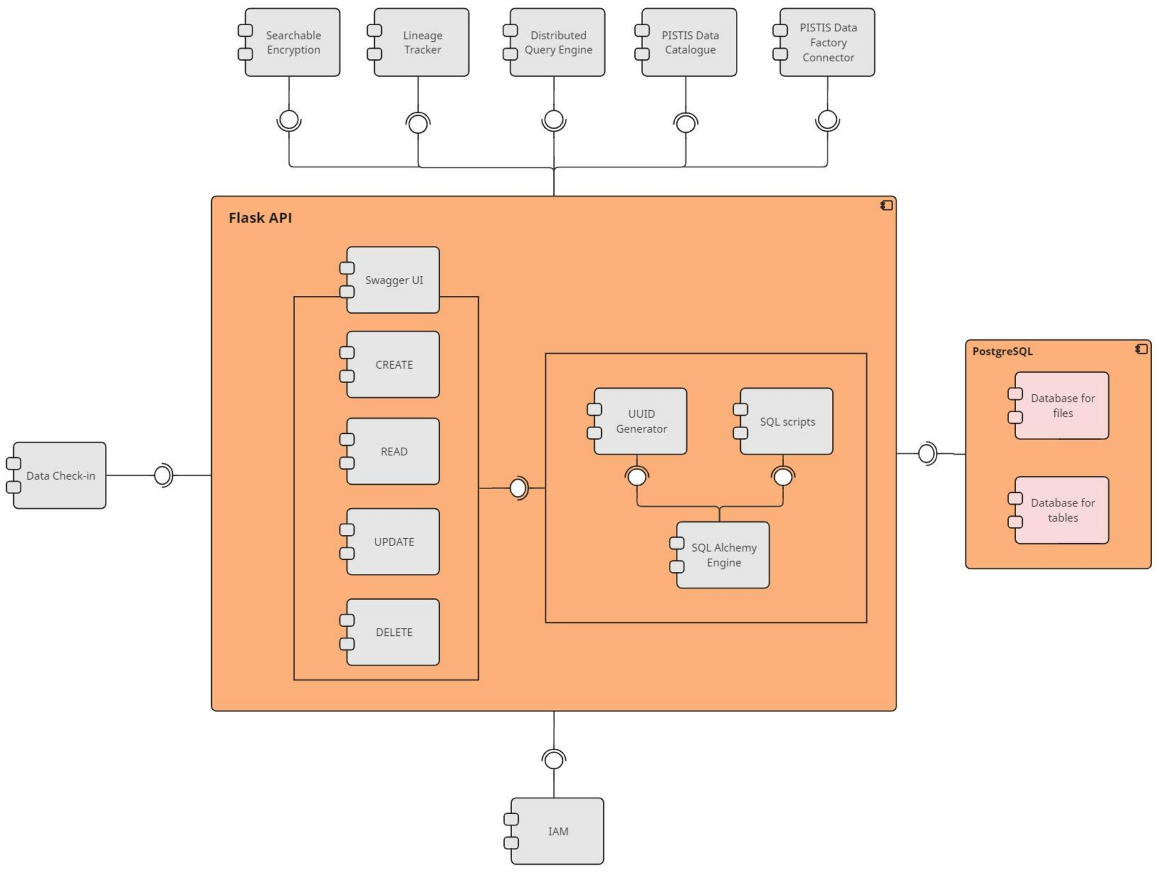


Figure 27: Factory Data Storage Internal Architecture

3.2.7 MOCK-UPS AND SCREENSHOTS

This is a backend service, and has no UI

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 will be employed to quickly obtain a list of matches. Subsequently, the list of potential matches yielded by the LSH methods will be further evaluated and combined with those returned by the Distributed Data Catalogue.

Finally, the merged list will be cross-checked with the Policy Engine that will be part of the Keycloack to decide if the users have access rights to the results. Subsequently, this list will be fed to a pretrained ML-model that will re-rank it in order to give prominence to the most relevant matches.

4.1.2 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, scikit-lean, tensorflow, pytorch etc.

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	In Progr.	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	In Progr.	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's Main Elements and Internal Architecture

The Distributed Query Engine is divided into three major subcomponents: the Distributed Query Service, the Locality Sensitive Hasing 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 dataser 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.

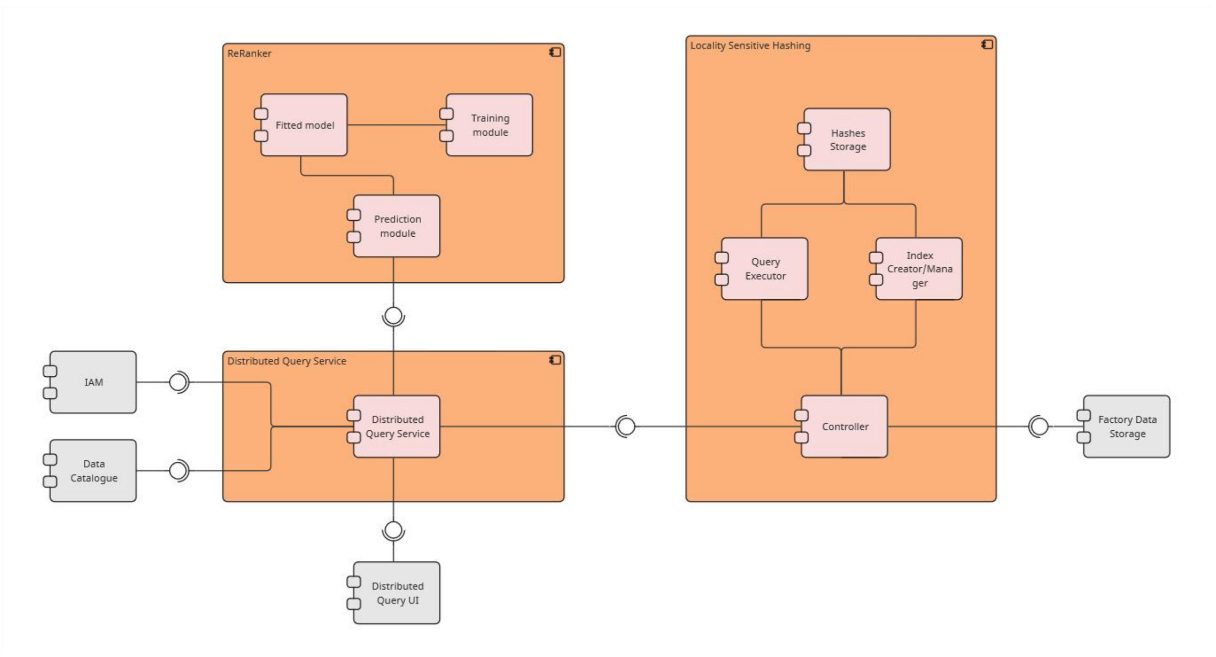
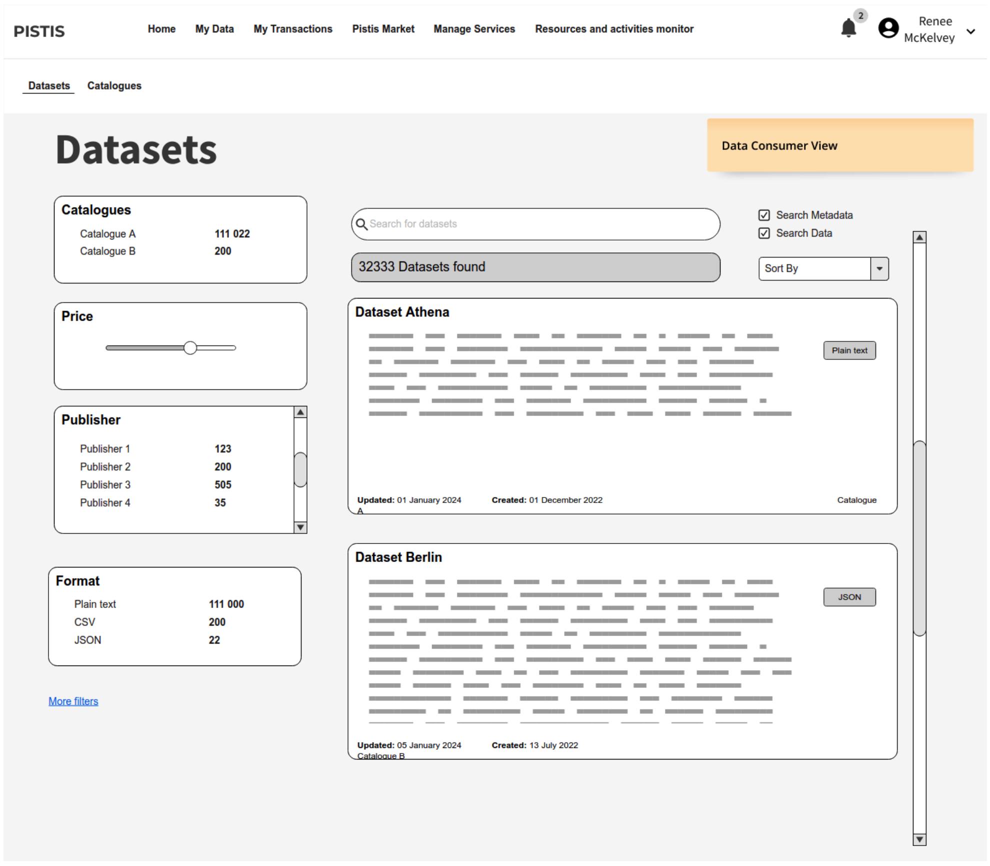


Figure 28: Distributed Query Engine Internal Architecture

4.1.7 Mock-ups and Screenshots



Catalogues

Catalogue A	111 022
Catalogue B	200

Price

Publisher

Publisher 1	123
Publisher 2	200
Publisher 3	505
Publisher 4	35

Format

Plain text	111 000
CSV	200
JSON	22

32333 Datasets found

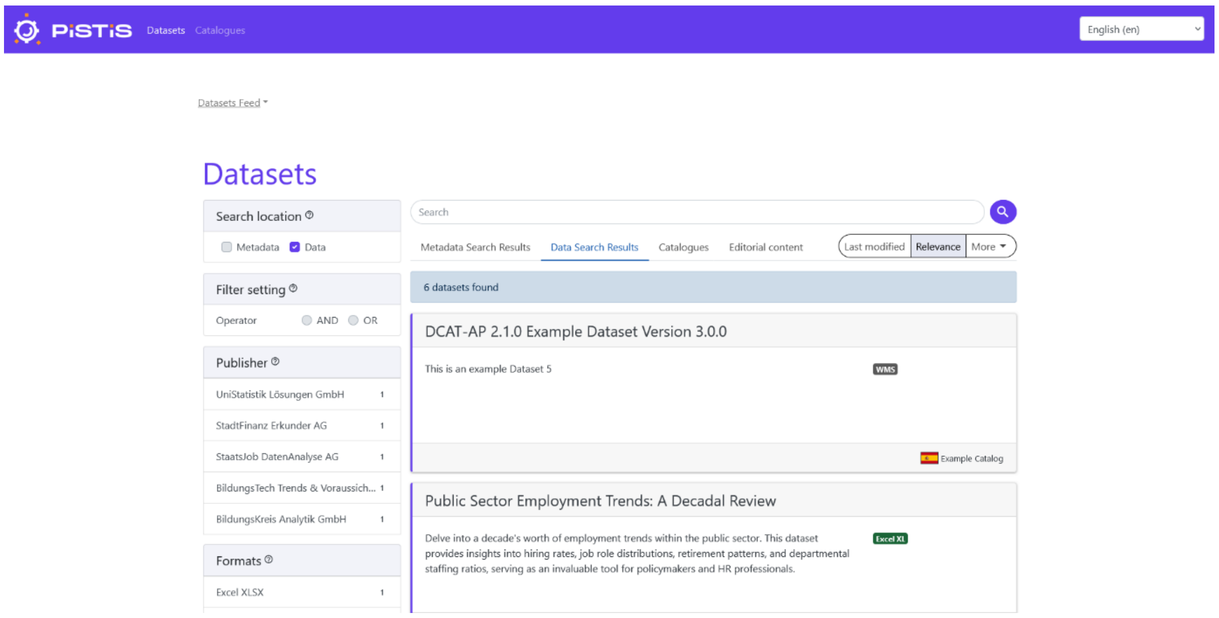
Dataset Athena

Updated: 01 January 2024 **Created:** 01 December 2022 Catalogue A

Dataset Berlin

Updated: 05 January 2024 **Created:** 13 July 2022 Catalogue B

Figure 29: Mock-up for searching datasets using the GUI



Datasets

Search location

[ ] Metadata [✓ Data]

Filter setting

Operator ○ AND ○ OR

Publisher

UniStatistik Lösungen GmbH	1
StadtFinanz Erkunder AG	1
StaatsJob DatenAnalyse AG	1
BildungsTech Trends & Voraussich...	1
BildungsKreis Analytik GmbH	1

Formats

Excel XLSX	1

6 datasets found

DCAT-AP 2.1.0 Example Dataset Version 3.0.0

This is an example Dataset 5 — WMS

Example Catalog

Public Sector Employment Trends: A Decadal Review

Delve into a decade's worth of employment trends within the public sector. This dataset provides insights into hiring rates, job role distributions, retirement patterns, and departmental staffing ratios, serving as an invaluable tool for policymakers and HR professionals. — Excel XL

Figure 30: Screenshot for searching datasets using the GUI (under development)

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 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.

5.1.3 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	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.12
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	Upcoming	PISTIS.OUS.12
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	Upcoming	PISTIS.OUS.12
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.12
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	Upcoming	PISTIS.OUS.12
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	Upcoming	PISTIS.OUS.12
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	Upcoming	PISTIS.OUS.12
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	Upcoming	PISTIS.OUS.12

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's Main Elements and Internal Architecture

The main elements comprising the PISTIS Data Factory Connector 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 assets'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

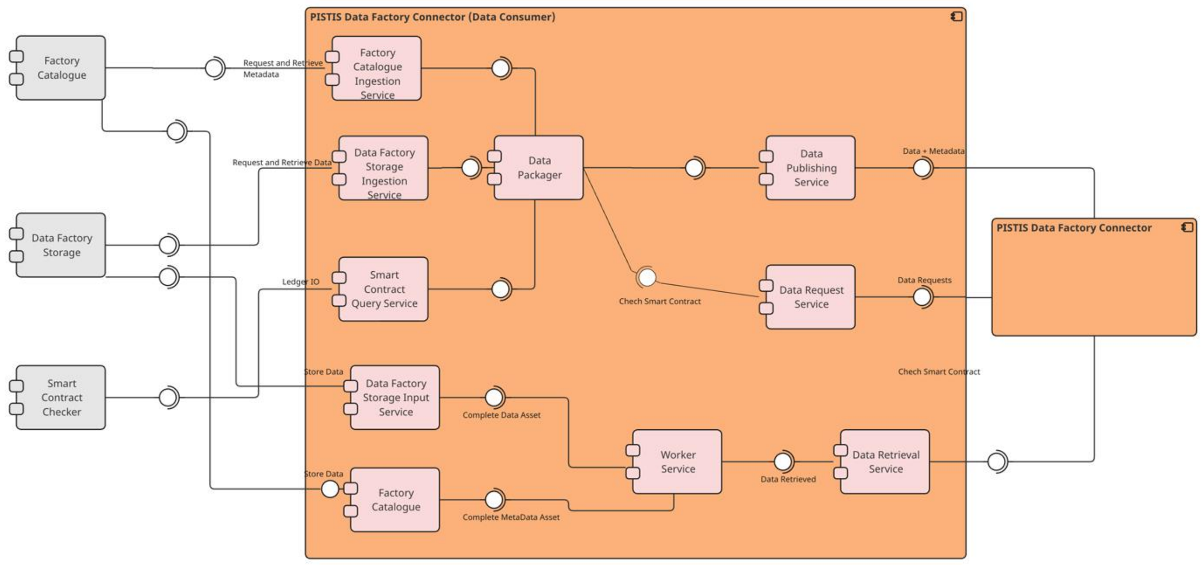


Figure 31: PISTIS Factory Connector's Internal Architecture

5.1.7 Mock-ups and Screenshots

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

5.2 Smart Contract Checker

5.2.1 Component Description

The Smart Contract Checker within the PISTIS platform is a component designed to ensure the integrity and validity of transactions on the network. Smart Contract Checker's checking policy is based on two pillars that collectively uphold the security, authenticity, and compliance of operations on the platform:

- Checking the authenticity of transactions: They originate from a valid, authenticated, and registered member of the PISTIS Platform. In other words, there is a need to check if the smart contract operations are valid and performed by an authorized user by checking all the signatures. Two checks will be performed: the first one is if the user has valid verifiable credentials (stored in the wallet), and the second one is if the user is a valid and authenticated member of the PISTIS platform.
- Ensuring there is no violation in future data transactions: This is done in the context of functional logic. For instance, by checking the common privacy protection profile and the exposed sensitive columns or the existing balance of money prior to data trading.

The Smart Contract Checker serves both the organizations and the PISTIS framework.

The Smart Contract Checker will be implemented as a rule-based engine that will be filled with a sample of fixed rules for the Alpha version. During the demonstrators implementation, and based on the outputs of the tasks that are developing the legal terms that should accompany the contracts, these rules will be updated, while at the end of the project the component will be able to accommodate any other rules deemed necessary by the operators of the PISTIS ecosystem.

5.2.2 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.

5.2.3 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 functionality	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	be sure that	I can be sure	Alpha	smart	Done	PISTIS

	Provider, Data Consumer	before data sharing that do not violate and policy (with UBITECH's rules)	that no issues will be arise		contract check functionality		.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	In Progress	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	In Progress	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	In Progress	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.
FR_02	Smart Contract Checker supports triggering GDPR check component.	UC_4, UC_5	One of the supported violation checks is done by triggering the GDPR check component.

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's Main Elements and Internal 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.

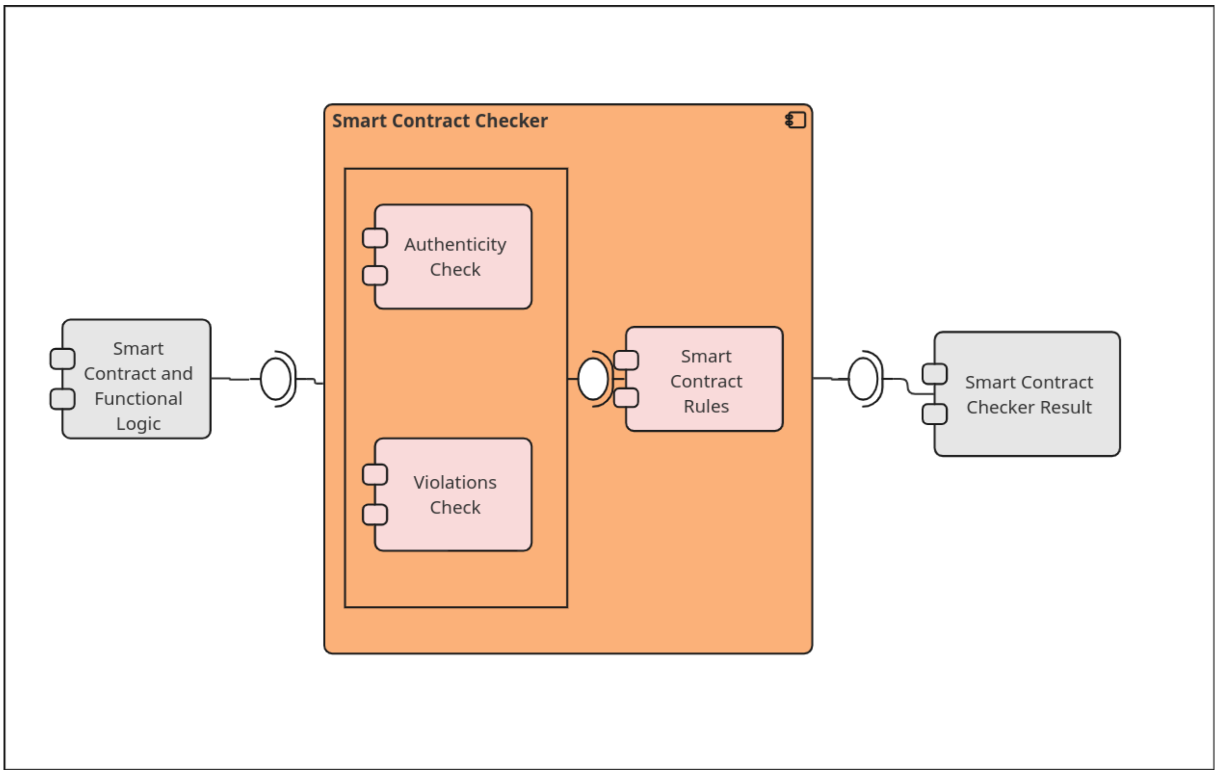


Figure 32: Smart Contract Checker High Level Architecture

5.2.7 Mock-ups and Screenshots

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

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 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

6.1.3 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, artifacts) 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	Upcom ing	PISTIS. SOUS. 02
UC_08	PISTIS Platform administrator	select multiple artefacts of the repo	I can download them	Beta	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	Upcom ing	PISTIS. SOUS. 02
UC_10	PISTIS Platform administrator	select multiple artefacts of the repo	I can delete them	Beta	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	Beta	Selected artefacts deleted	Upcoming	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/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's Main Elements and Internal 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

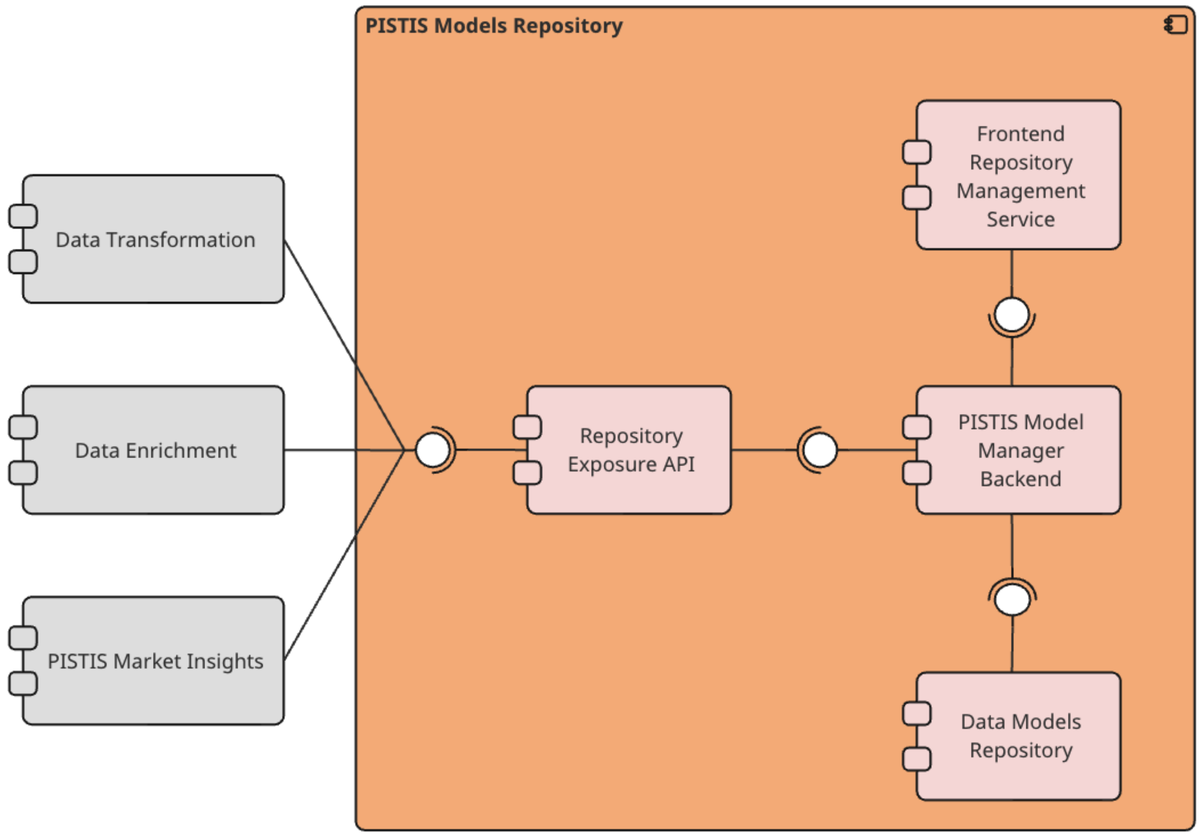
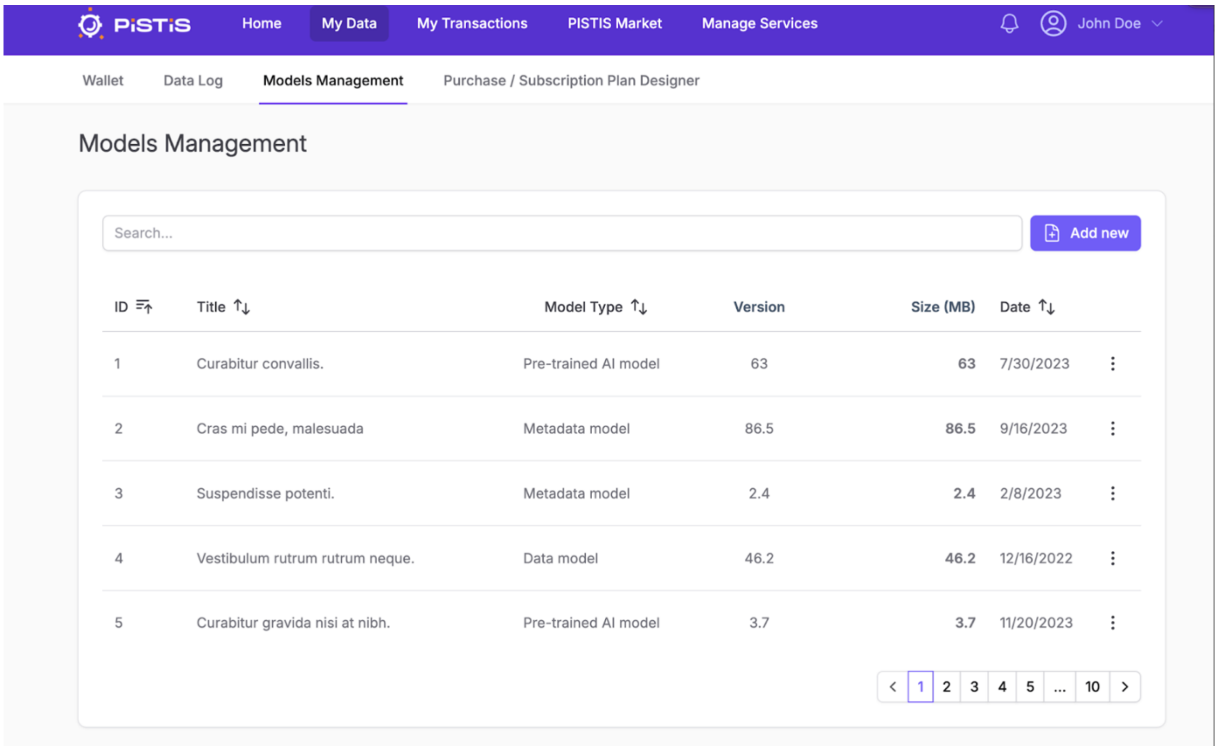


Figure 33: PISTIS Model Repository Internal Architecture

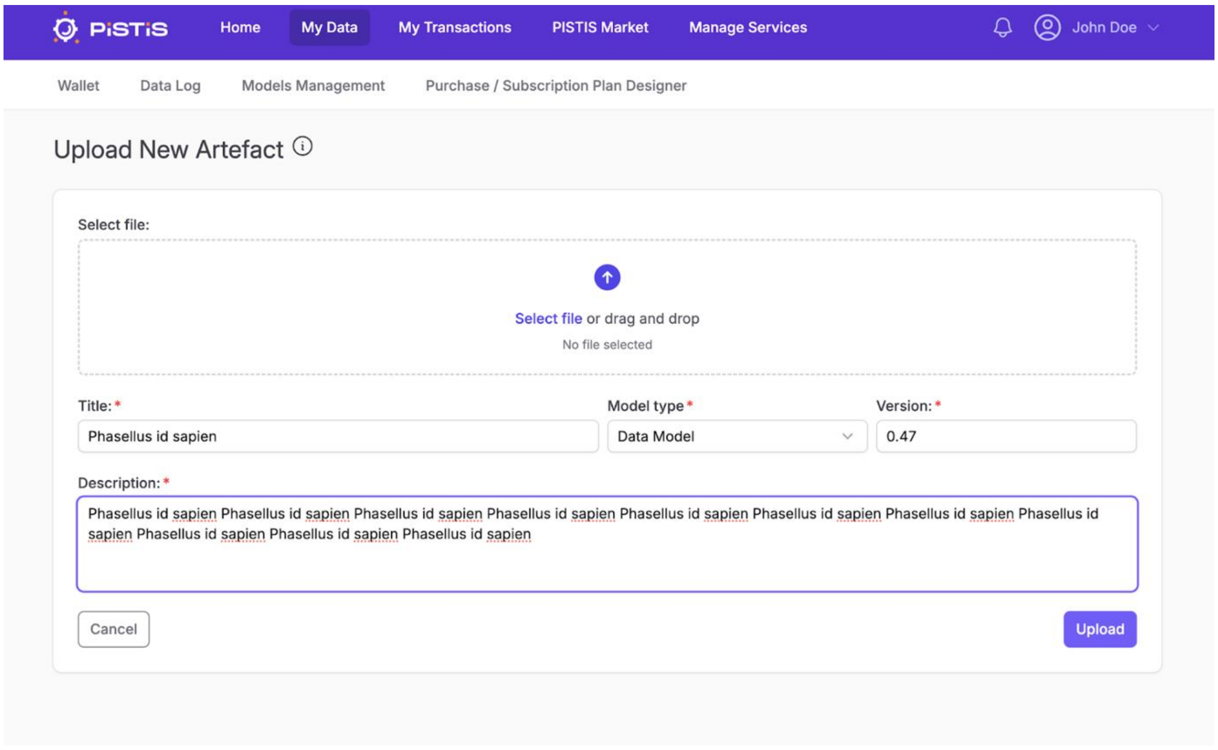
6.1.7 Mock-ups and Screenshots



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 34: PISTIS Models Repository – Models Management



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 35: PISTIS Models Repository – Upload of New Artefact

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 Technology Background

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

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

As it is shown in Figure 36, 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.

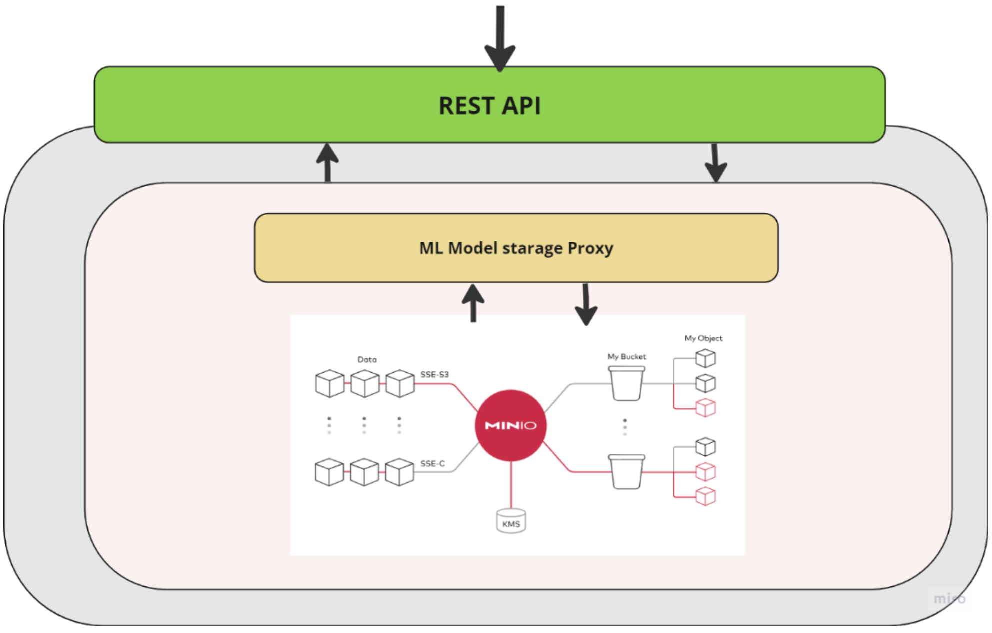
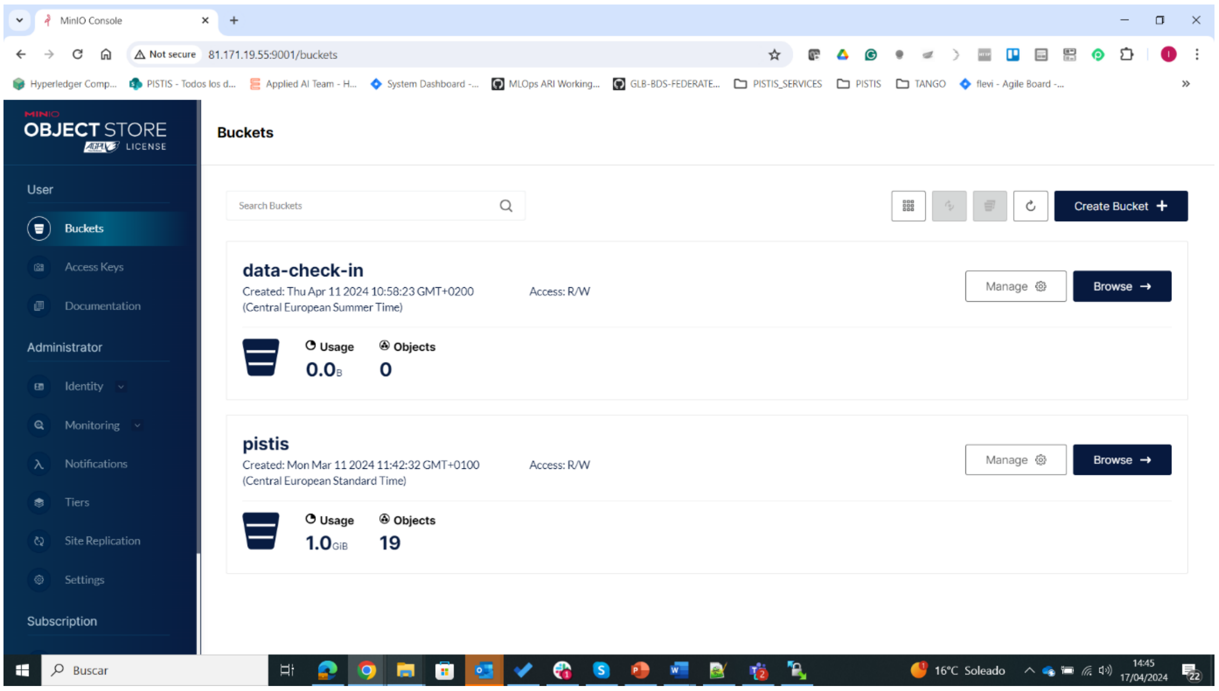


Figure 36: ML Model Repo Architecture

6.2.7 Mock-ups and Screenshots

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 37.



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 37: MinIO GUI

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 Technology Background

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

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.	Upcoming	PISTIS.OUS.03, PISTIS.OUS.07

6.3.4 Functional Requirements

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

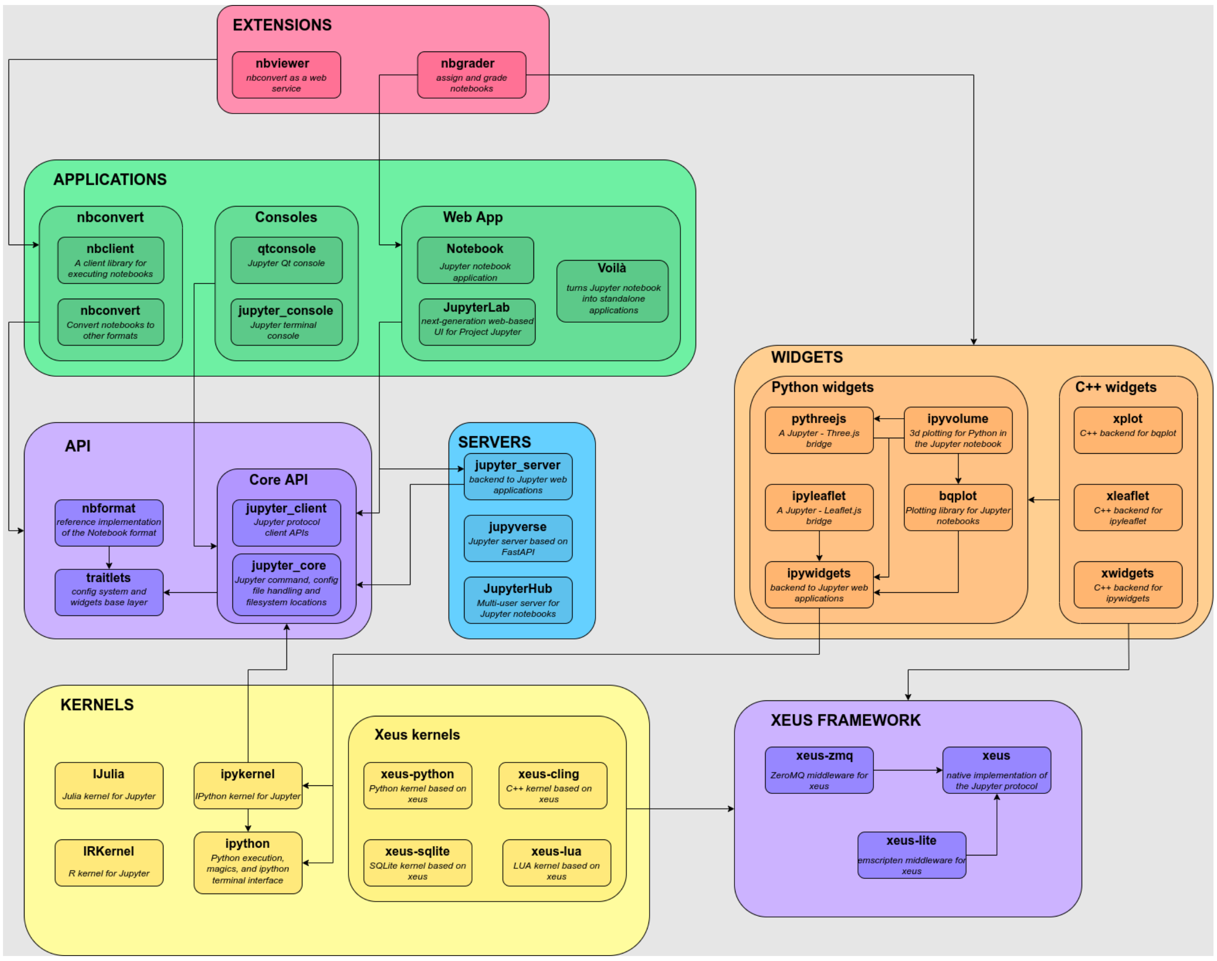
ID	Description	Related Use Cases	Comments
FR_01	Create, edit, and sharing computational AI models	UC1, UC2, UC3, UC4 and UC5	

6.3.5 Non-Functional Requirements

There are no non-functional requirements at the moment.

6.3.6 Component's Main Elements and Internal 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 38, 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
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
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
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 38: AI Model Editor Architecture[5]

6.3.7 Mock-ups and Screenshots

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.

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

A screenshot of Jupyter Lab UI is shown in Figure 39.

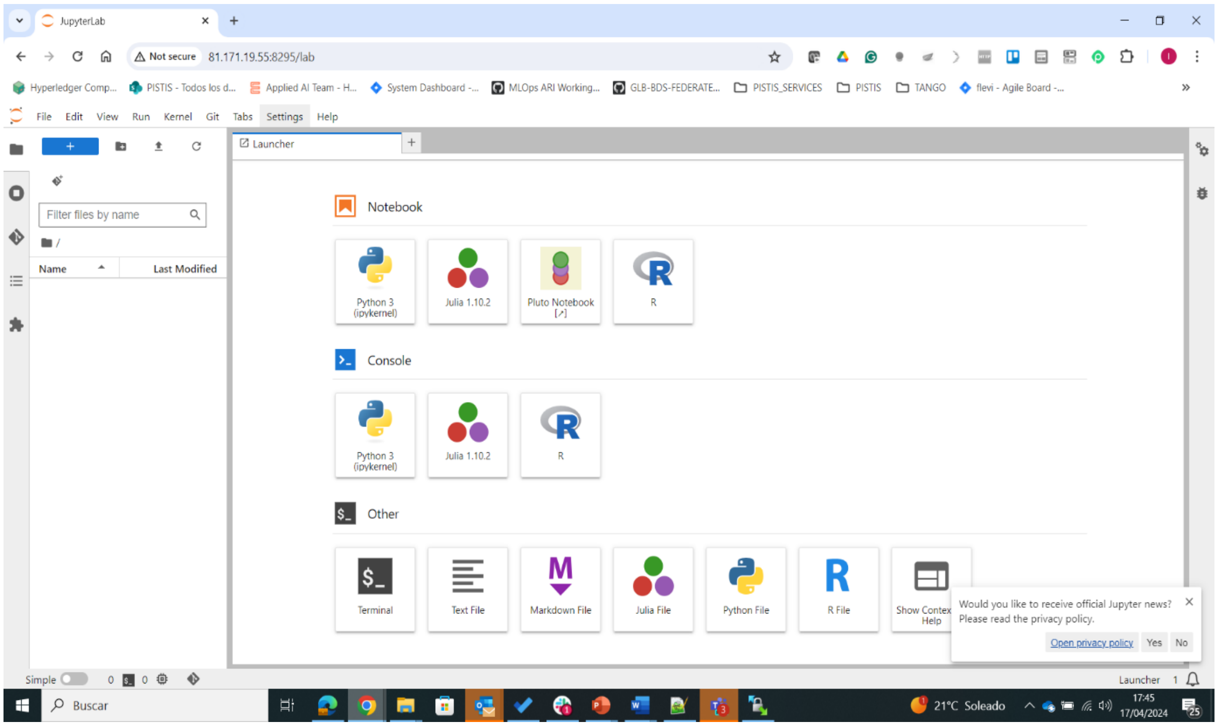


Figure 39: Jupyter Lab UI.

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

The anonymizer 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 anonymization process by selecting different anonymization 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 anonymization. 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 anonymization 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.

Finally, the anonymizer supports the use of differential privacy to allow for generalised insights to be derived from data in such a way that reverse engineering to re-identify individuals within a dataset is not possible.

7.1.2 Technology Background

Anonymizer Technical Overview

The anonymizer consists of a few modularized components. The k-Anonymity provided by the anonymizer will be supported by a dockerized java application using a springboot API. The rest of the anonymization functionality (deletion, data masking, location anonymization, 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 anonymization functions from a single API.

Anonymizer Technical Details

K-Anonymity

The main functionality of the anonymizer is handled by the ARX data anonymization java library. The anonymizer 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 springboot. 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 yellow, results in purple).

```
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`.

Generalisation

Differential Privacy

DiffPrivLib is a python library supported by IBM that supports differential privacy in machine learning models. It is responsible for injecting mathematical noise into machine learning results to prevent re-identification of individuals by reverse engineering using standard differential privacy mechanisms such as Laplace distributions to determine appropriate amounts of noise. A privacy budget is also used to limit the amount of information that can be gained via queries and other means of data access. Each query expends privacy budget until the privacy budget reaches its maximum at which point further queries will be denied. By default, no privacy budget is used meaning that based on sensitivity requirements.

DiffPrivLib currently supports a subset of machine learning algorithms and pre-processing using similar syntax to the scikit-learn python library. The library supports algorithms for clustering, classification, regression, dimensionality reduction and pre-processing.

Based on sensitivity requirements provided by the user and other PISTIS components the correct algorithm will be selected and configured.

Anonymizer Location API Technical Details

The location handler is a python-based component responsible for the anonymization 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 anonymization. To provide an internal API to pass data between the Anonymizer 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.

7.1.3 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	UPCO MING

					mathematical and statistical properties as the original real dataset	

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's Main Elements and Internal Architecture

The figure below shows the internal architecture of the anonymiser. Its main elements include support for all the functionality described in Section 7.1.2.

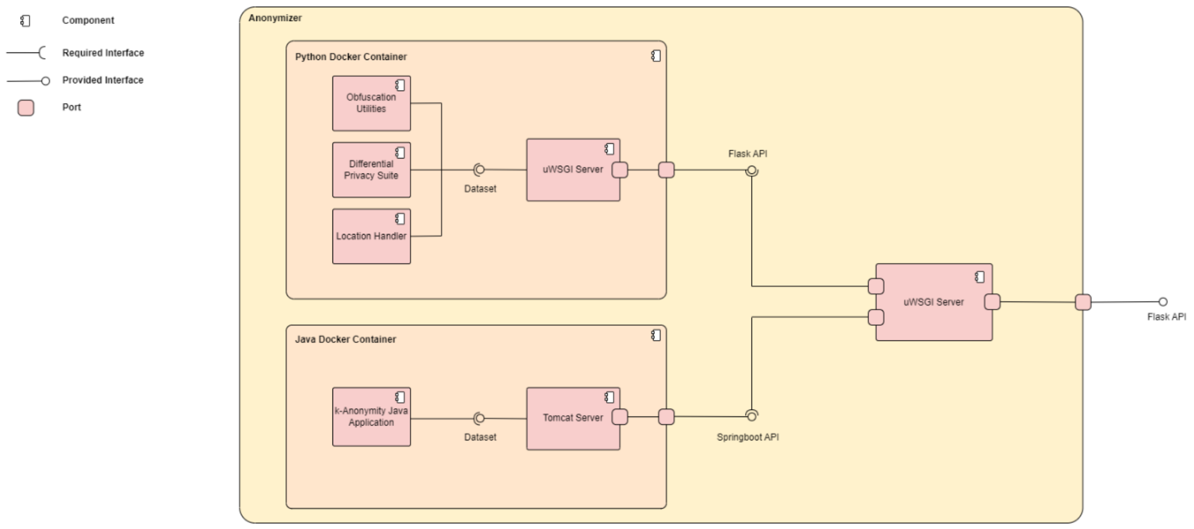
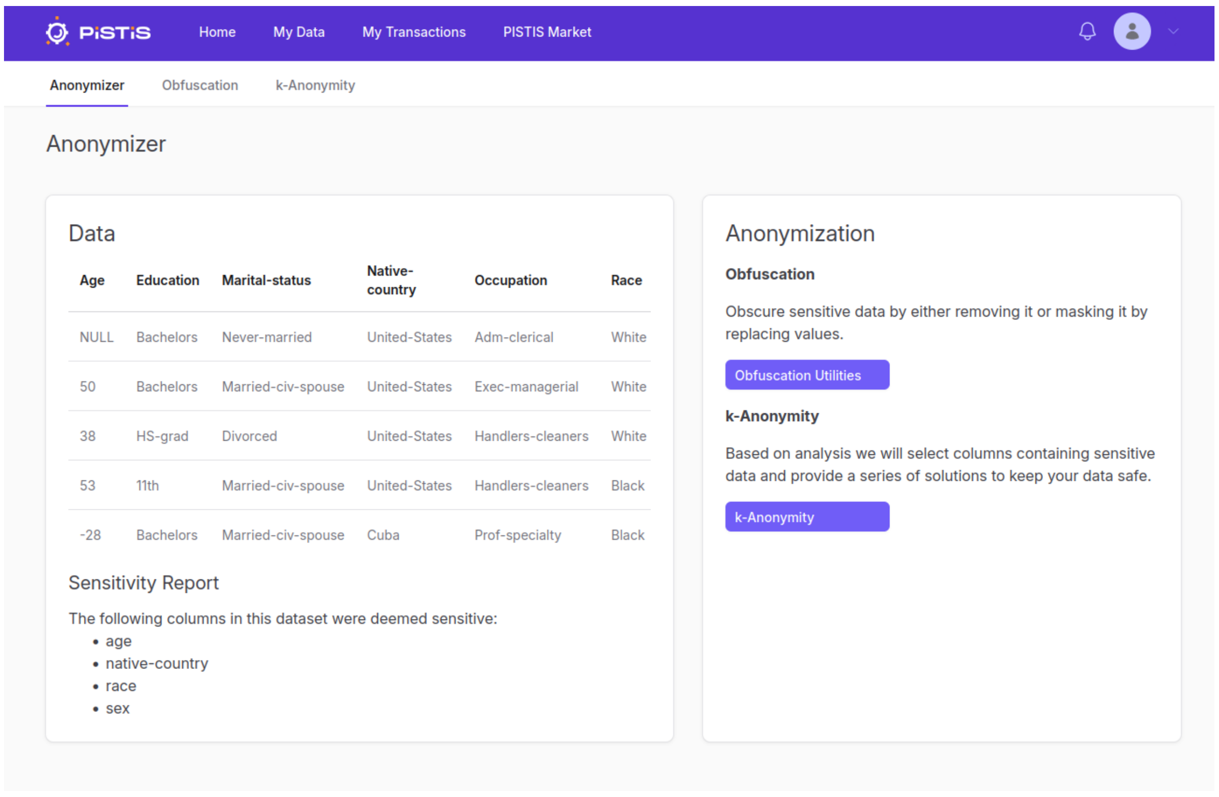


Figure 40: Component's Internal Architecture

7.1.7 Screenshots



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

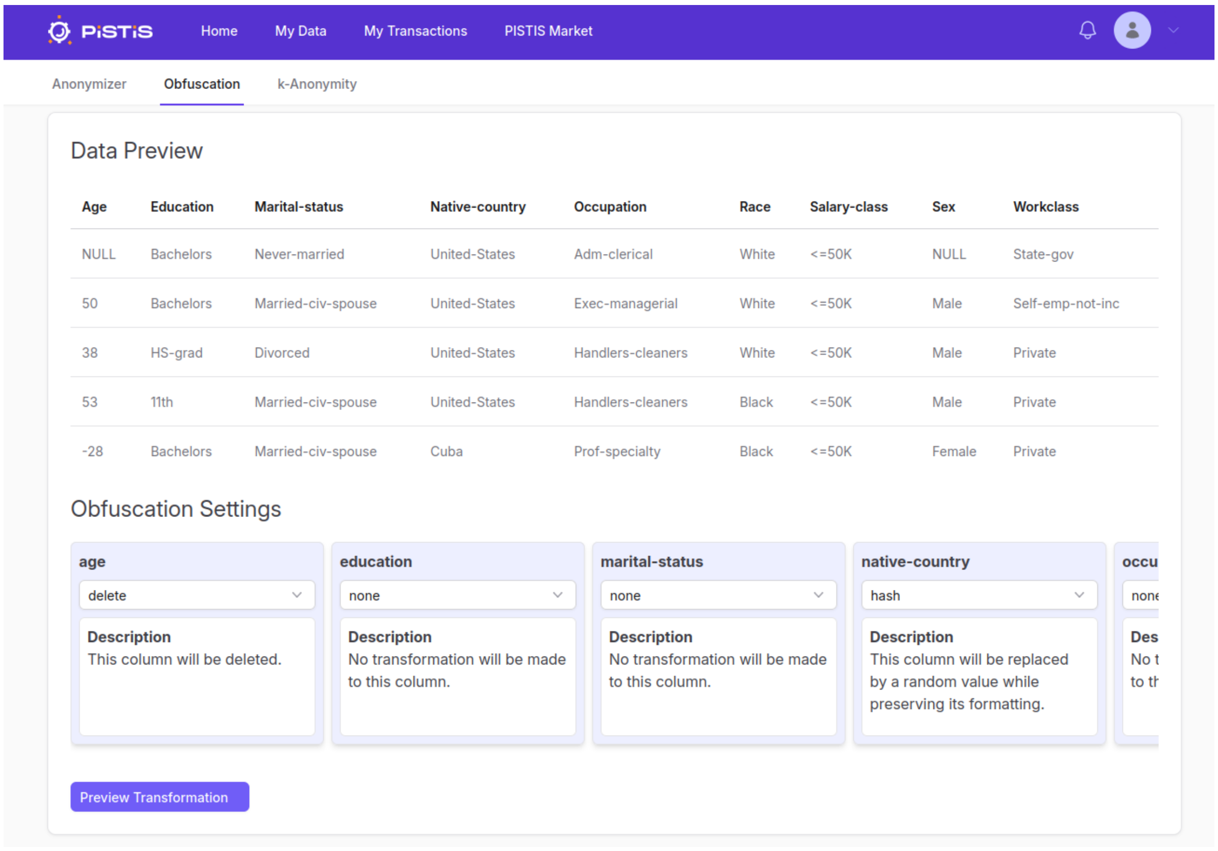
Obfuscation

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 41: 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: delete
Description
This column will be deleted.

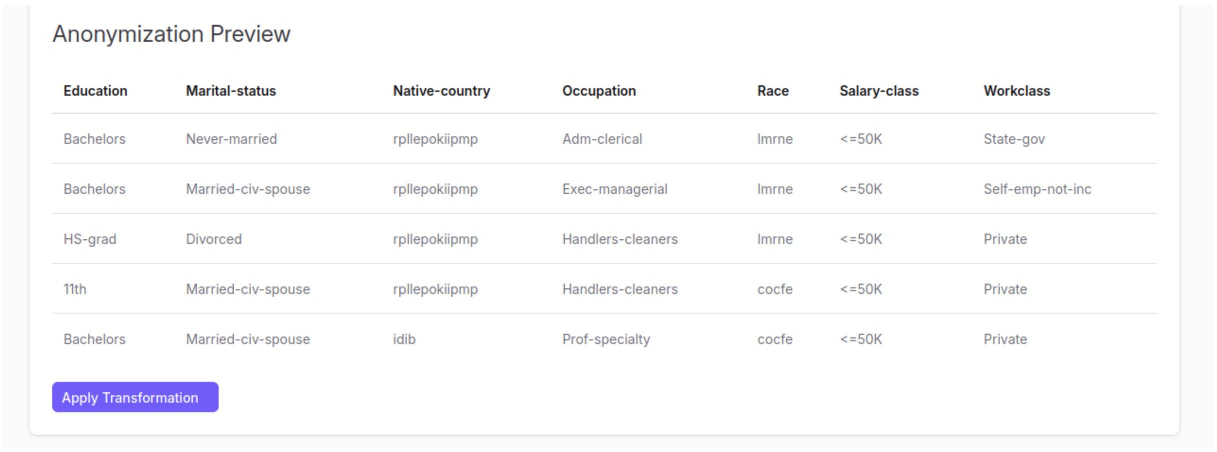
education: none
Description
No transformation will be made to this column.

marital-status: none
Description
No transformation will be made to this column.

native-country: hash
Description
This column will be replaced by a random value while preserving its formatting.

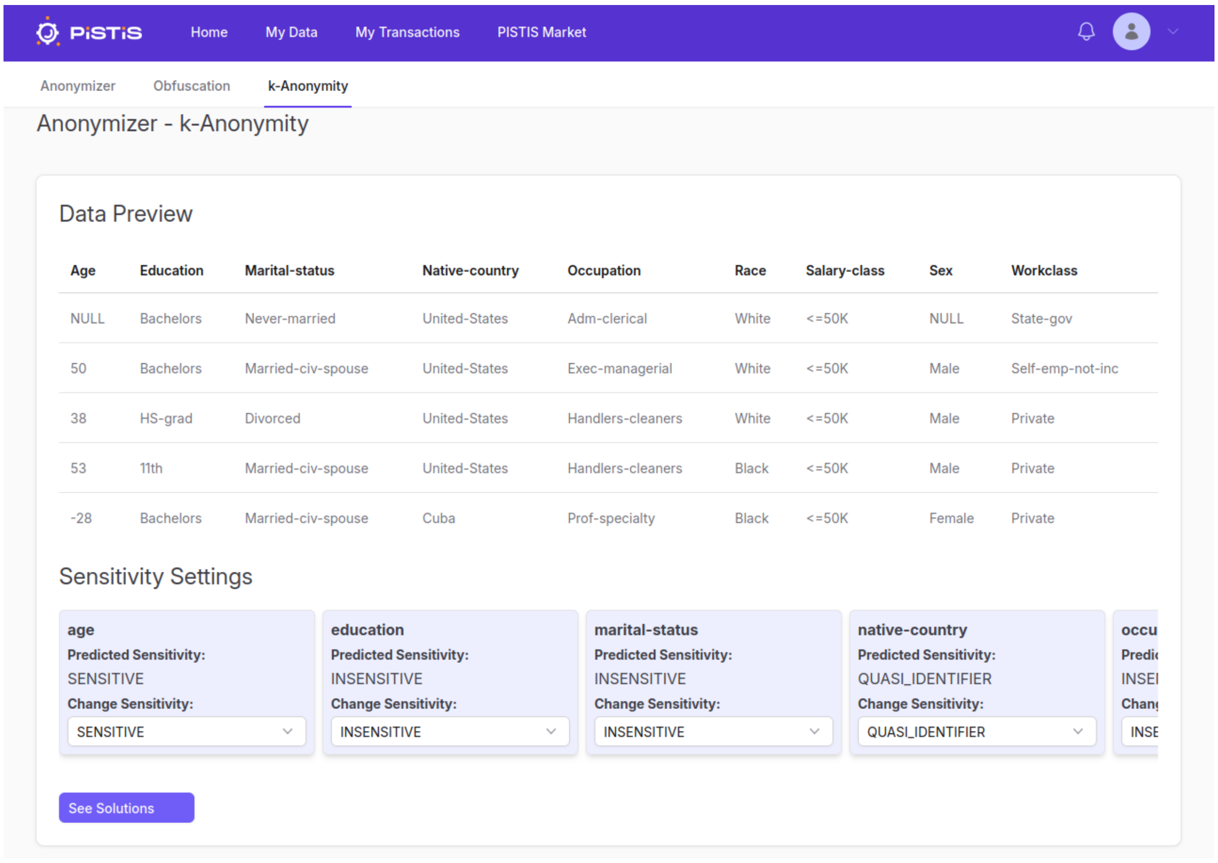
occu: none
Des
No t
to th

Figure 42: Anonymiser Obfuscation Utilities


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

Figure 43: Obfuscation Anonymiser Preview



Anonymizer - k-Anonymity

Data Preview

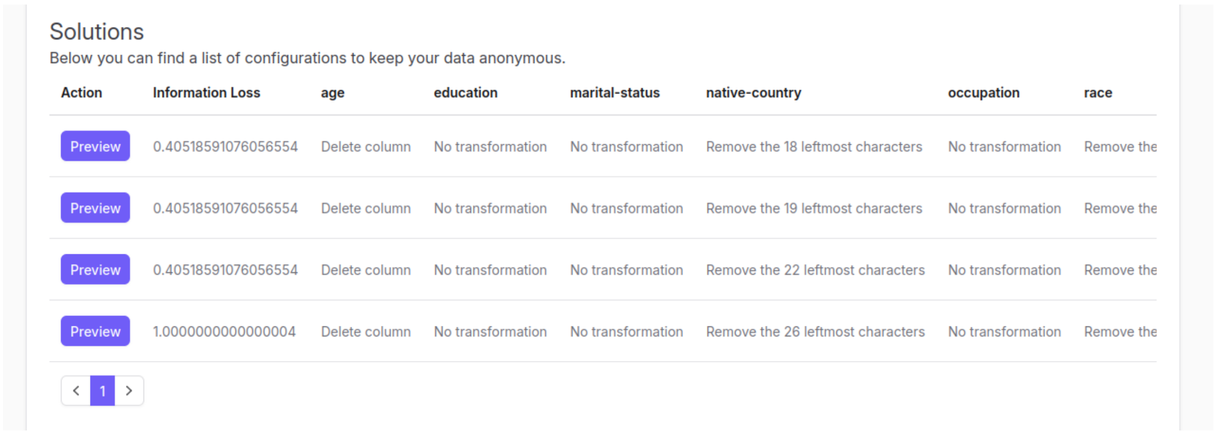
Age	Education	Marital-status	Native-country	Occupation	Race	Salary-class	Sex	Workclass
NULL	Bachelors	Never-married	United-States	Adm-clerical	White	<=50K	NULL	State-gov
50	Bachelors	Married-civ-spouse	United-States	Exec-managerial	White	<=50K	Male	Self-emp-not-inc
38	HS-grad	Divorced	United-States	Handlers-cleaners	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	education	marital-status	native-country	occu
Predicted Sensitivity:	SENSITIVE	INSENSITIVE	INSENSITIVE	QUASI_IDENTIFIER	INSE
Change Sensitivity:	SENSITIVE	INSENSITIVE	INSENSITIVE	QUASI_IDENTIFIER	INSE

See Solutions

Figure 44: Anonymiser K-Anonymity

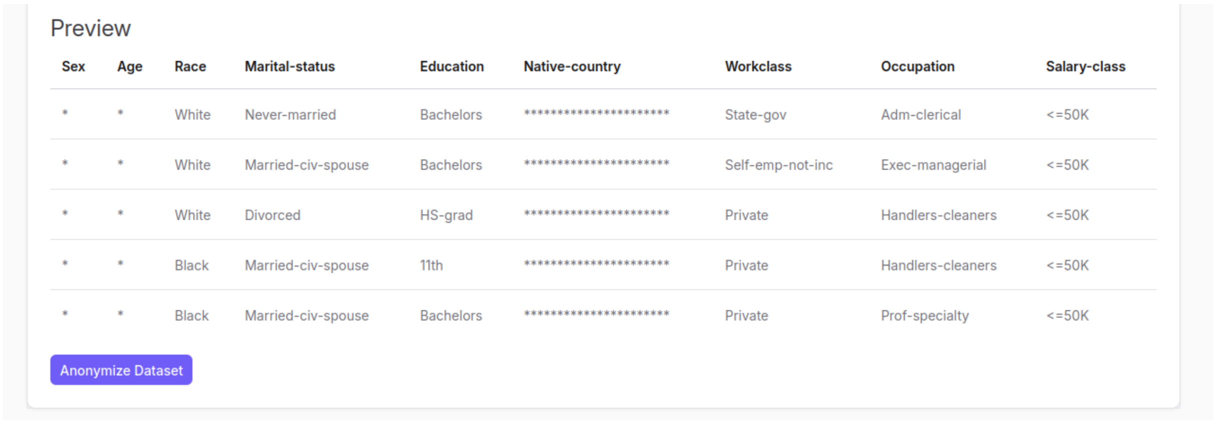


Solutions

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

Figure 45: Anonymiser K-Anonymity Solutions



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

Figure 46: Preview of Dataset after K-Anonymity Solution is selected

7.2 LINEAGE TRACKER

7.2.1 COMPONENT DESCRIPTION

The Lineage Tracker documents operations performed on a dataset as well as by whom and when the operations were performed. By constructing the dataset's lineage representing its version history, the Lineage Tracker provides the user with transparency and access to previous dataset versions.

The Lineage Tracker offers API endpoints to document the operations performed on a specific dataset and to receive lineage information in a structured format. When a user creates, reads, updates or deletes a dataset, this information is converted into RDF-format (Linked Data) using the W3C PROV-Ontology and saved in a triple store. Each dataset's lineage graph is then continuously extended with every operation. Users can then retrieve lineage information by querying the API for a dataset's history, lineage, family tree, and a user's history.

Lastly, the Lineage Tracker offers a user-friendly user interface (UI), allowing the user to view a dataset's family tree, lineage, and history of operations. More specifically, the UI displays this information in the form of a tree and table, showing the user every operation performed on each dataset.

7.2.2 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 Boostrap for responsive CSS styling.

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 (Backend) In progress (UI)	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 (Backend) In progress (UI)	PISTIS. OUS. 08
UC_03	Data Consumer	Verify who is accessing/ manipulatin g my dataset.	I have visibility over who is accessing my data.	Alpha (Backend) Beta (UI)	Data Provider can view all Data Consumer accessing dataset.	Done (Backend) In progress (UI)	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'S MAIN ELEMENTS AND INTERNAL ARCHITECTURE

Brief description of internal component's elements and the architecture of the component

Per Figure 47Figure 1 below, the Lineage Tracker consists of the Flask REST API, Openlink Virtuoso triple store, and Vue.js UI.

The Flask REST API is used to create operation documentation and information retrieval API endpoints that can be called by the Factory Data Storage and end users, respectively. Furthermore, the Flask REST API coordinates between requests from the Factory Data Storage and end users, and the Openlink Virtuoso triple store. When a dataset is created, updated, deleted, or read by a user, the Factory Data Storage makes corresponding changes to the dataset and then calls the Flask REST API to record this lineage information. The python-prov library is then used to convert this lineage data to RDF, which is stored in the Virtuoso triple store as a graph. Whenever a dataset is created, updated, deleted, or read, this corresponding graph is also updated.

The Openlink Virtuoso triple store is used to store lineage information. Whenever a dataset is created, updated, deleted, or read by a user, the Factory Data Storage stores this resulting RDF data in the Openlink Virtuoso triple store. Furthermore, whenever a user requests lineage data, this data is retrieved from the Openlink Virtuoso triple store using a SPARQL query.

Vue.js is used to visualize the lineage information in a user-friendly UI. When a user wants to view a dataset's lineage information, they can do so using the Vue.js UI, which in turn requests this data from the Flask REST API. The Vue.js UI then displays this information in the form of a tree and table, allowing the user to view a dataset's family tree, lineage, and history of operations.

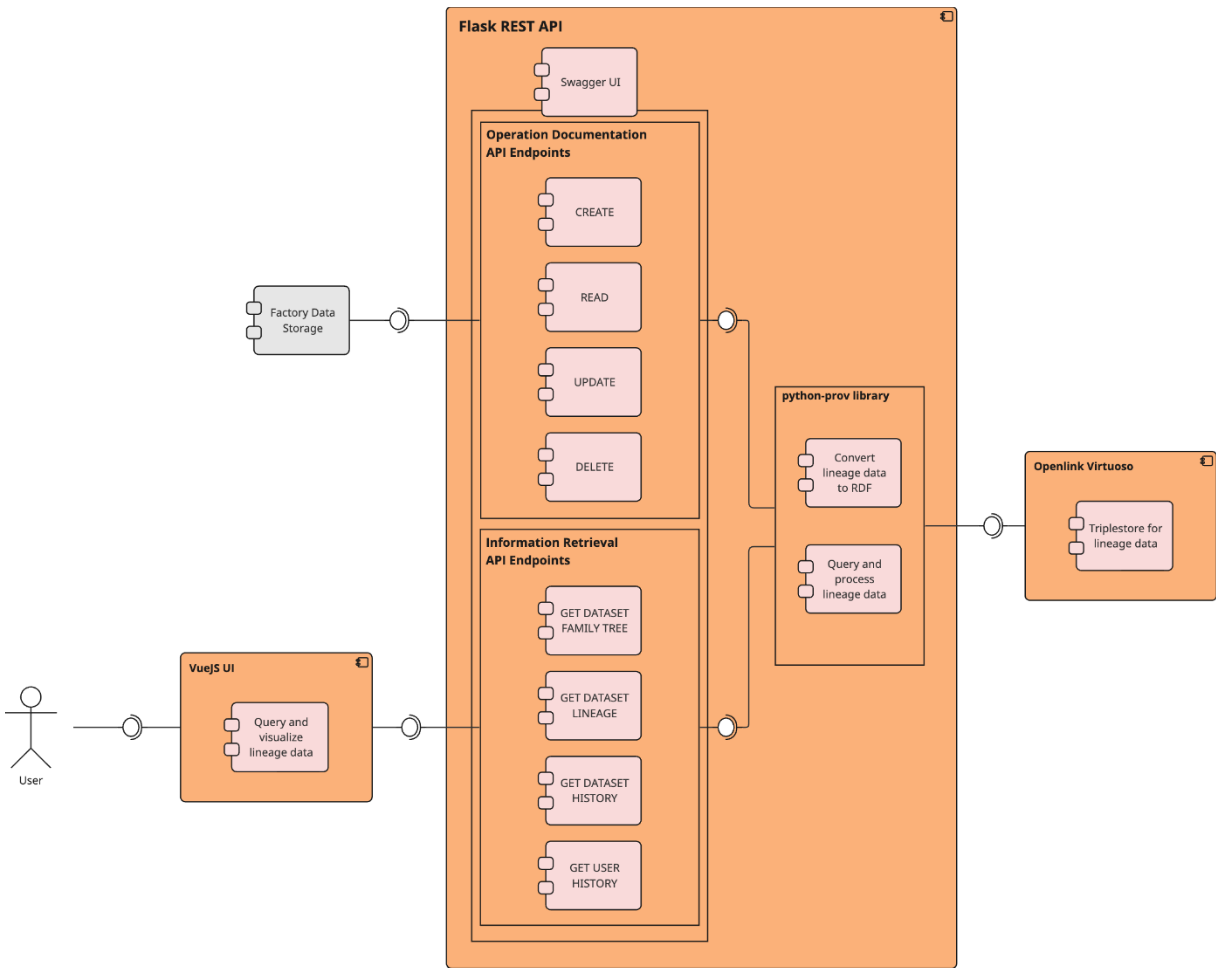
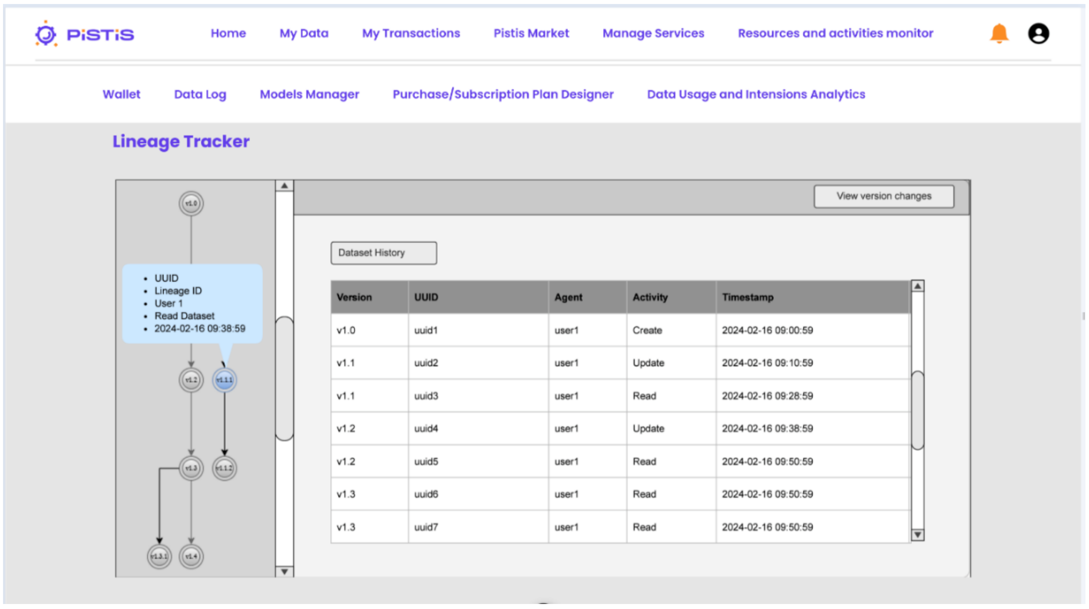


Figure 47: Lineage Tracker Architecture Diagram

7.2.7 Mock-ups and Screenshots

The first mock-up below provides information on a dataset's lineage information. Specifically, the mock-up shows a dataset's family tree in the form of a graph as well as the dataset's lineage history in the form of a table.

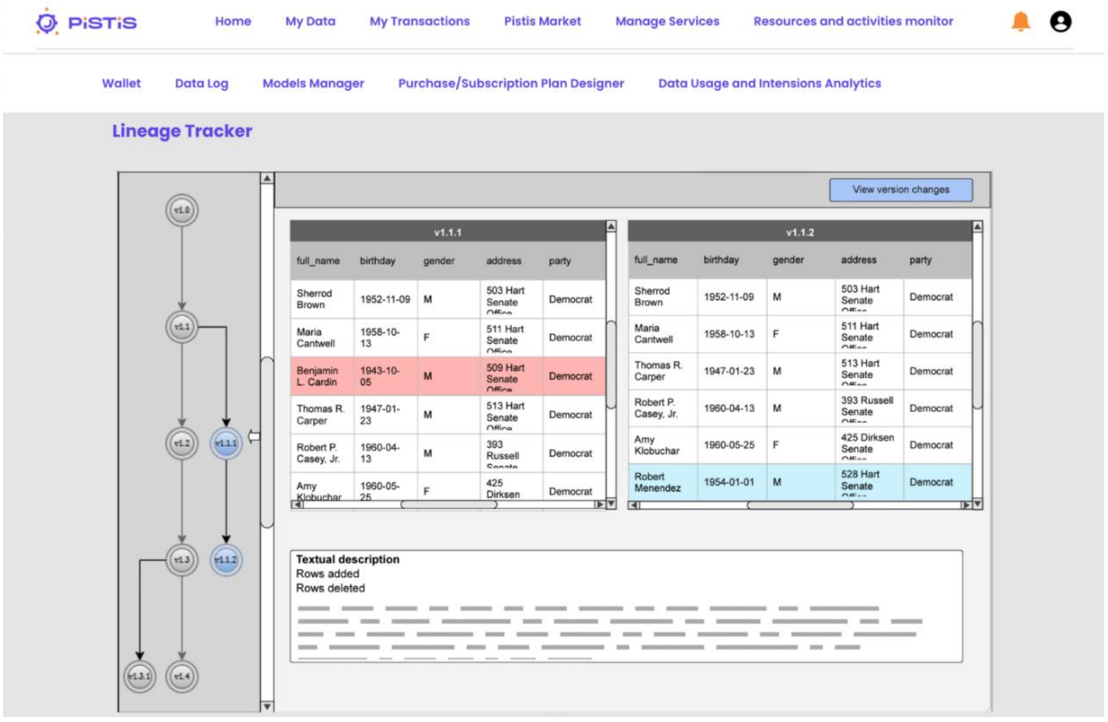


- UUID
- Lineage ID
- User 1
- Read Dataset
- 2024-02-16 09:38:59

Version	UUID	Agent	Activity	Timestamp
v1.0	uuid1	user1	Create	2024-02-16 09:00:59
v1.1	uuid2	user1	Update	2024-02-16 09:10:59
v1.1	uuid3	user1	Read	2024-02-16 09:28:59
v1.2	uuid4	user1	Update	2024-02-16 09:38:59
v1.2	uuid5	user1	Read	2024-02-16 09:50:59
v1.3	uuid6	user1	Read	2024-02-16 09:50:59
v1.3	uuid7	user1	Read	2024-02-16 09:50:59

Figure 48: View Dataset Lineage

The second mock-up below compares the differences between two datasets in a family tree. Specifically, the mock-up shows a dataset's family tree in the form of a graph and highlights the differences between two datasets in the form of a table.



v1.1.1				
full_name	birthday	gender	address	party
Sherrod Brown	1952-11-09	M	503 Hart Senate	Democrat
Maria Cantwell	1958-10-13	F	511 Hart Senate	Democrat
Benjamin L. Cardin	1943-10-05	M	509 Hart Senate	Democrat
Thomas R. Carper	1947-01-23	M	513 Hart Senate	Democrat
Robert P. Casey, Jr.	1960-04-13	M	393 Russell	Democrat
Amy Klobuchar	1960-05-25	F	425 Dirksen	Democrat

v1.1.2				
full_name	birthday	gender	address	party
Sherrod Brown	1952-11-09	M	503 Hart Senate	Democrat
Maria Cantwell	1958-10-13	F	511 Hart Senate	Democrat
Thomas R. Carper	1947-01-23	M	513 Hart Senate	Democrat
Robert P. Casey, Jr.	1960-04-13	M	393 Russell Senate	Democrat
Amy Klobuchar	1960-05-25	F	425 Dirksen Senate	Democrat
Robert Menendez	1954-01-01	M	528 Hart Senate	Democrat

Textual description
Rows added
Rows deleted

Figure 49: View Dataset Version Changes

7.3 GDPR CHECKER

7.3.1 COMPONENT DESCRIPTION

The GDPR Checker component is responsible for ensuring compliance with the EU General Data Protection Regulation (GDPR). In a nutshell, the GDPR Checker not only checks compliance with respect to anonymity but also against the privacy profile (e.g., anonymity, linkability, observability, etc.) of the user. More specifically, this privacy profile will be checked against the data that is going to be uploaded to the PISTIS platform and can be exchanged through the platform to ensure compliance. The necessary level of privacy depends on the sensitivity and nature of the data, as well as the application requirements that produce this data. These privacy profiles, based on the characteristics of the data, can capture the required level of privacy protection. These measures (privacy profiles) are expressed as GDPR-alignment policies. Policies need to be in the form of "if type of data X and type of data Y is exposed, then this can have an impact on the privacy property k (e.g., anonymity) of the user."

Considering this, the core functionality of the GDPR Checker is to generate a type of smart contract that can act as certification of GDPR compliance (including the user's consent) for further processing of the data. The component will consider the labels of the data (from the Data Quality Assessment component) and the privacy profiles (to be defined by the organization's legal entity) and will provide either a certification of compliance with GDPR or static mitigation suggestions to the PISTIS Platform UI.

GDPR Checker will be implemented as a rule-based engine that will be filled with a sample of our own rules for the Alpha version. This implementation is not the final version of it since the actual rules will be provided by our partners that are leading the legal issues.

7.3.2 TECHNOLOGY BACKGROUND

The GDPR Checker component utilizes modern web technologies to provide compliance solutions in line with the EU General Data Protection Regulation (GDPR). 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 GDPR 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.

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., anonymisations)	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 the official rules)	make the necessary actions (e.g., anonymisations)	Beta	GDPR compliance report	In Progress	PISTIS.OUS.4 & PISTIS.OUS.7
US_04	Data Consumer	rest assured that the data I acquired are GDPR compliant (according to the official rules)	no legal issues occur	Beta	GDPR compliance report	In Progress	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	GDPR compliance certificated stored in the Public Ledger as a proof of compliance.	UC_1, UC_2	Other PISTIS components can check the datasets compliance.
FR_03	If dataset is not GDPR compliant potential mitigation measures in the form of suggestions will be provided to the used (Data Provider)	UC_1, UC_2	The Data Provider will be responsible to perform any of the suggestions (e.g., use the Anonymizer)

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 compliance and the GDPR

		certificate should be stored in a way this could not be altered.

7.3.6 Component's Main Elements and Internal Architecture

The GDPR Checker tool is architected to ensure that data handling processes comply with GDPR standards through a structured evaluation system. It integrates two key modules for initial data analysis: the 'Sensitive Data Identification' module and the 'Privacy Policy Identification' module. The former scans for sensitive personal information, while the latter examines associated privacy policies for GDPR compliance requirements. The outputs from these modules are then analysed by the 'Check Compliance' module, which assesses conformity to GDPR norms. Depending on the results of this assessment, the tool either issues a 'Certificate of Compliance' or provides 'Static Mitigation Recommendations' to address any compliance shortfalls. This architecture enables a systematic approach to validate and ensure adherence to GDPR mandates.

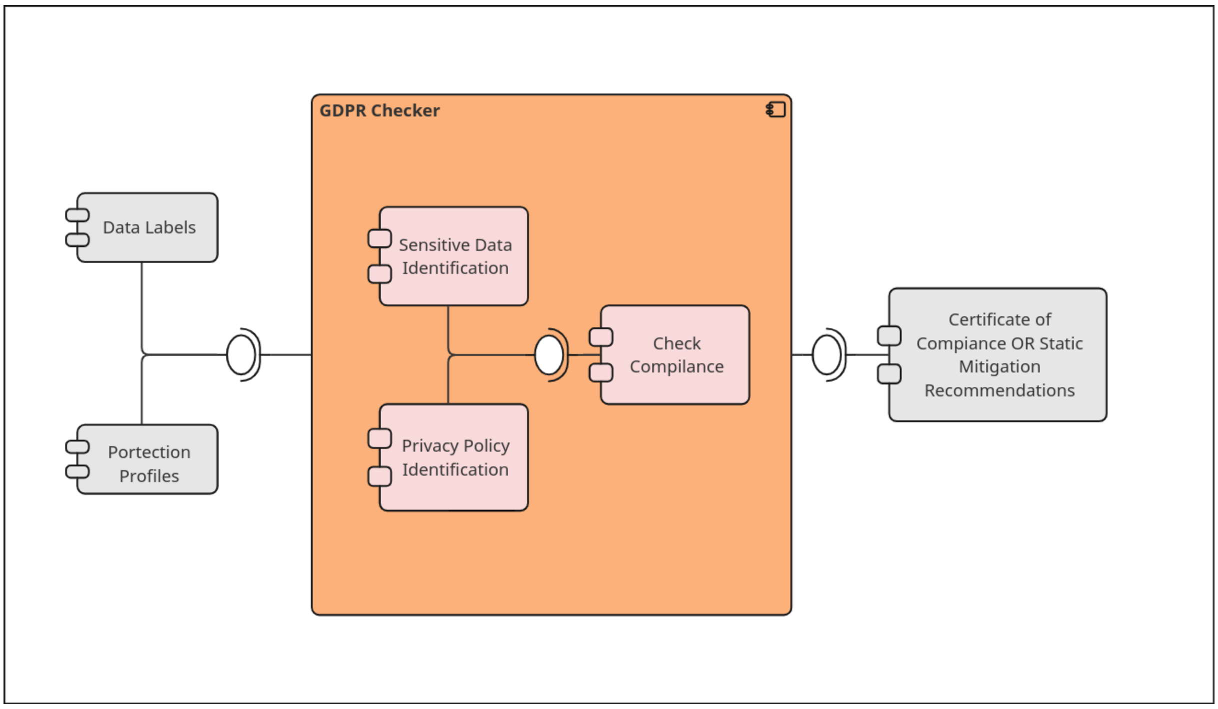

GDPR Checker
Data Labels
Portection Profiles
Sensitive Data Identification
Privacy Policy Identification
Check Compilance
Certificate of Compiance OR Static Mitigation Recommendations

Figure 50: GDPR Checker High Level Architecture

7.3.7 Mock-ups and Screenshots

The component doesn't have a GUI.

7.4 SEARCHABLE ENCRYPTION

7.4.1 COMPONENT DESCRIPTION

The Searchable Encryption (SE) component is a fundamental aspect of the PISTIS platform and enables data users to search over the encrypted data (or indexes to data). In other words, it allows users to perform searches on data that has been encrypted, ensuring that sensitive information is protected from unauthorized access. Indeed, SE not only protects the data privacy of Data Providers but also enables data users to search over the encrypted data. However, it assumes that the user sending the search query owns the decryption key and that the sender must know the identity of the user querying the data in order to encrypt using the corresponding encryption key. This raises the question of how to achieve this since the encrypted data are shared between several receivers. To remedy this, we can borrow ideas from attribute-based encryption (ABE), where only participants with the appropriate permissions and the corresponding ABE (matching) attribute private keys can decrypt and view the encrypted data (indexes). Concretely, the secret decryption key of the encrypted indexes is related to a set of attributes in some fashion, for which holds that if there is a subset of attributes that consists of at least t attributes that match the set of attributes associated with the secret keys, then the secret keys can be used to decrypt them.

In the context of PISTIS, the SE component will adopt a dynamic searchable encryption (DSE) scheme over encrypted indexes stored in the Blockchain and the Factory Data Storage. In addition, SE enables the exposure encrypted up to a certain level of granularity of the type of data but not the content of the data. For instance, we can identify the type of data based on keywords associated with that data. In other words, such encrypted indexes serve as pointers to the encrypted stored data. DSE supports efficient data or keyword modification via an update mechanism. On top of that, the mapping between the keywords and the encrypted indexes will be stored in the Blockchain.

Searchable Encryption is not going to be included in Alpha version of PISTIS. The following sections define what is needed and the whole architecture of this component shortly, but the implementation will take place in the Beta phase.

7.4.2 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.

7.4.3 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	In Progress	PISTIS.OUS.9
US_02	Data Provider	hide actual type of data	I can protect my privacy	Beta	Search results	In Progress	PISTIS.OUS.9
US_03	Data provider	only users with specific attributes to search my data	I can protect my privacy	Beta	Search results	In Progress	PISTIS.OUS.9
US_04	Data Provider	to dynamically update the encrypted keyword	the keyword to be more accurate upon changes	Beta	Search results	In Progress	PISTIS.OUS.9
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	In Progress	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	In Progress	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'S MAIN ELEMENTS AND INTERNAL ARCHITECTURE

Figure 51 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.

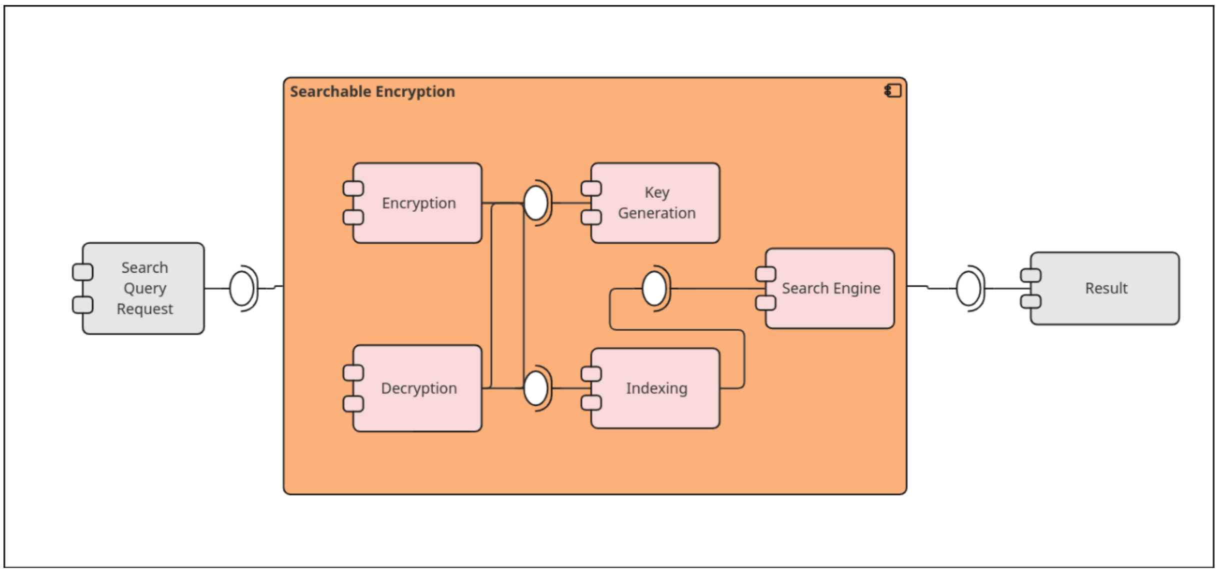


Figure 51: Searchable Encryption high-level Architecture

7.4.7 MOCK-UPS AND SCREENSHOTS

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

7.5 ENCRYPTION/DECRYPTION ENGINE

7.5.1 COMPONENT DESCRIPTION

The Encryption/Decryption engine within the PISTIS platform is a comprehensive and versatile tool designed to handle a wide array of encryption and decryption needs. This component not only supports standard encryption methodologies, such as symmetric and asymmetric encryption, but also extends its capabilities to ensure robust data protection and privacy across various aspects of the platform. Moreover, in the context of the SSI concept the key for encryption/decryption is associated with the Self-Sovereign Identity (SSI) of the user.

7.5.2 TECHNOLOGY BACKGROUND

The Encryption/Decryption Engine component is built using a combination of **Flask**, a lightweight **Python** web framework, and **C**. The core functionality of the engine, which involves the actual encryption and decryption of data, is implemented in C due to its efficiency and performance in executing low-level computations. C's ability to interact directly with system hardware and memory makes it ideal for implementing cryptographic algorithms that require high-speed processing and stringent security measures.

In addition to the Flask and C implementations, the component leverages EJBCA[6], a robust Public Key Infrastructure (PKI) certificate authority software package. EJBCA supports a variety of certificate authority functionalities and is used here to manage digital certificates and public-keys, ensuring secure data transactions. This PKI system enables the Encryption/Decryption Engine to support multiple CAs and different levels of CAs, facilitating the construction of a complete and versatile security infrastructure within a single instance of the software.

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.10
US_02	Data Provider	be able to encrypt datasets	I can protect data	Alpha	Encryption process	Done	PISTIS.OUS.1

[6] EJBCA Enterprise from PrimeKey, Website: https://www.primekey.com/products/software/ejbca-enterprise/

			confidentiality				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'S MAIN ELEMENTS AND INTERNAL 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.

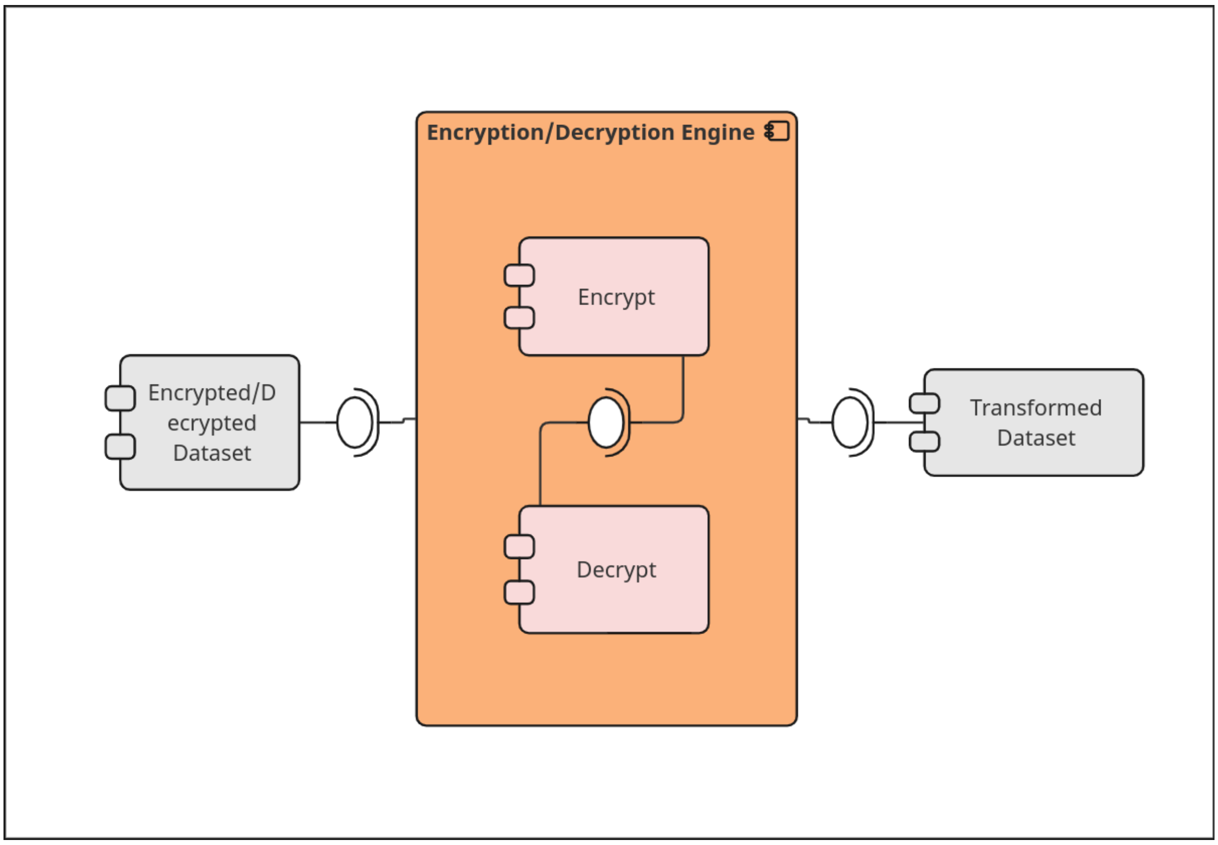


Figure 52: Encryption/Decryption Engine High Level Architecture

7.5.7 Mock-ups and Screenshots

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

7.6 Access Policy Editor

7.6.1 Component Description

Access Policy Editor is a component integrated with Keycloak[7] 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 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.

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

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 Technology Background

PISTIS Access Policy Editor component, is a web-based GUI realized with NuxtJS[8] 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.

7.6.3 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 access policies on a newly created asset	Can control access to the asset from the organization users	Alpha	Access to the asset is governed by the policies created by	Done	PISTIS.OUS.05

[8] https://nextjs.org

					the asset owner		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	Ongoing	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 he same organization

7.6.6 Component's Main Elements and Internal Architecture

The main elements of Access Policy Editor are:

- A standalone Web-based GUI to provide to the PISTIS Organisation Administrators with ease management of Users and Policies within their organization.
- 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.
- The Validation Service aiming to mock policies outputs for various inputs scenarios in order to allow testing defined policies efficiency before actually deploying them. The component will again use Keycloak available operations via API Adapter; however, it is mentioned as a separate component for clarity.
- 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.

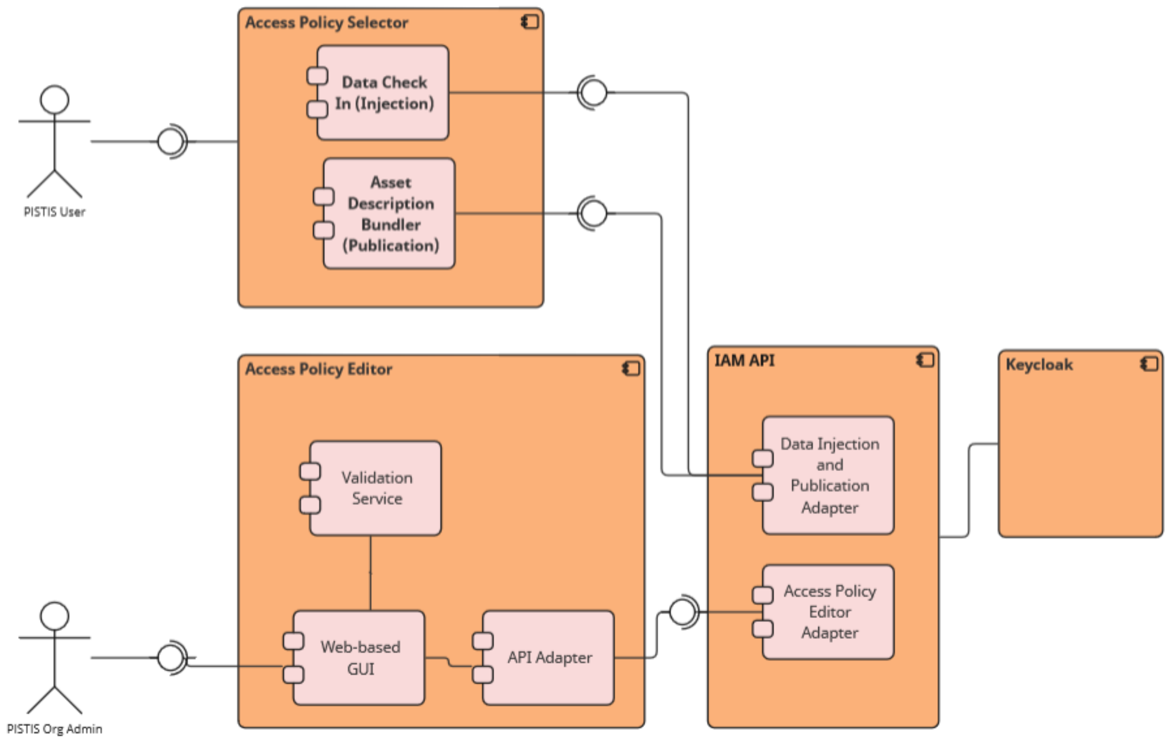
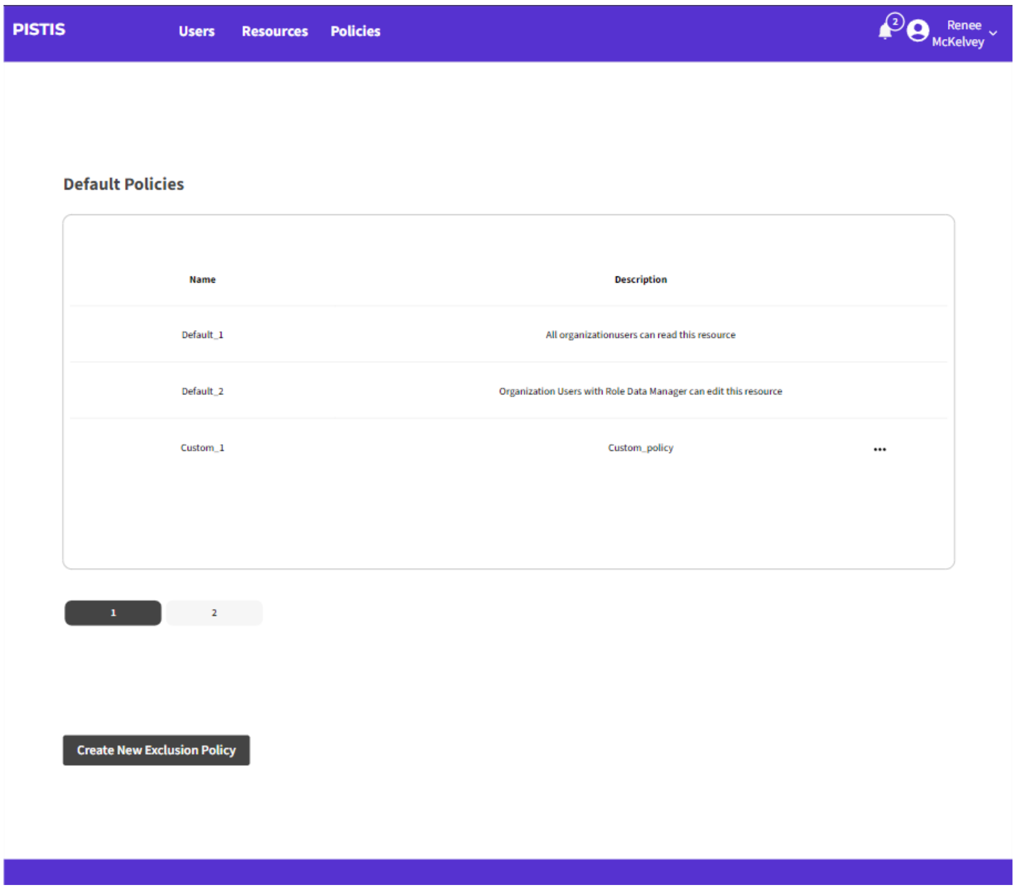

Access Policy Selector
Data Check In (Injection)
Asset Description Bundler (Publication)
PISTIS User
Access Policy Editor
Validation Service
Web-based GUI
API Adapter
PISTIS Org Admin
IAM API
Data Injection and Publication Adapter
Access Policy Editor Adapter
Keycloak

Figure 53: Access Policy Editor Internal Architecture

7.6.7 Mock-ups and Screenshots

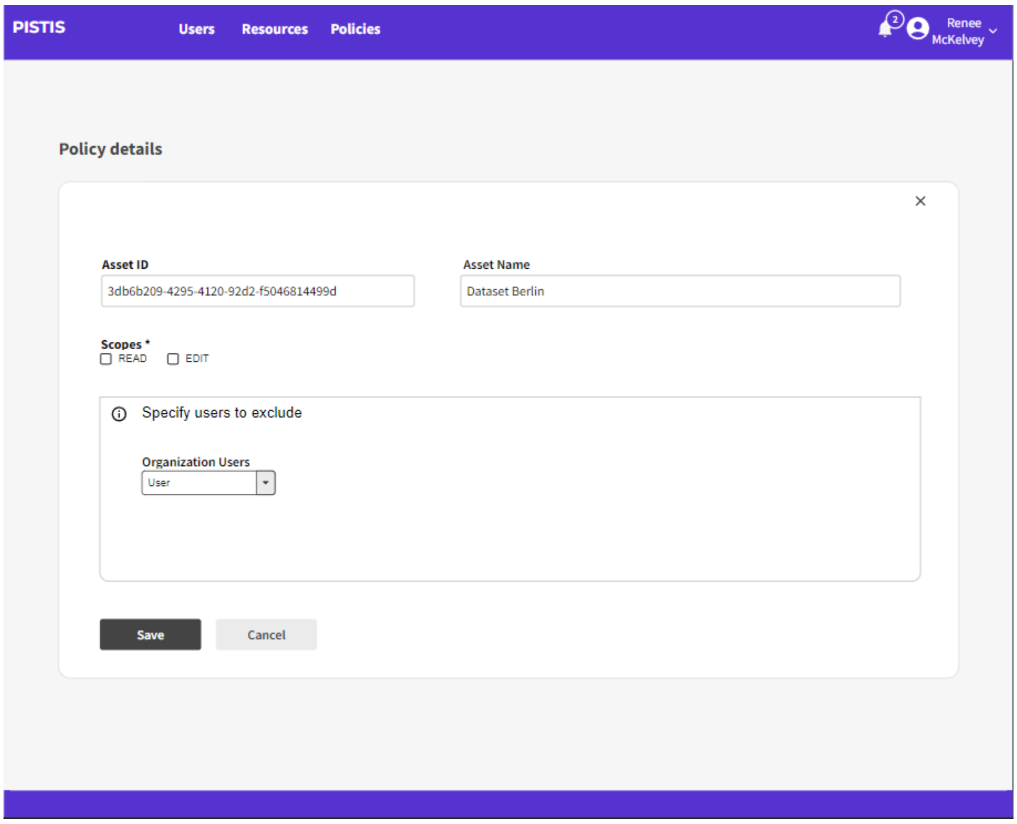
7.6.7.1 *Screens related to Data Asset Injection*



Default Policies

Name	Description
Default_1	All organizationusers can read this resource
Default_2	Organization Users with Role Data Manager can edit this resource
Custom_1	Custom_policy

Figure 54: Listing of access policies during data asset injection phase



Policy details

Asset ID	3db6b209-4295-4120-92d2-f5046814499d
Asset Name	Dataset Berlin

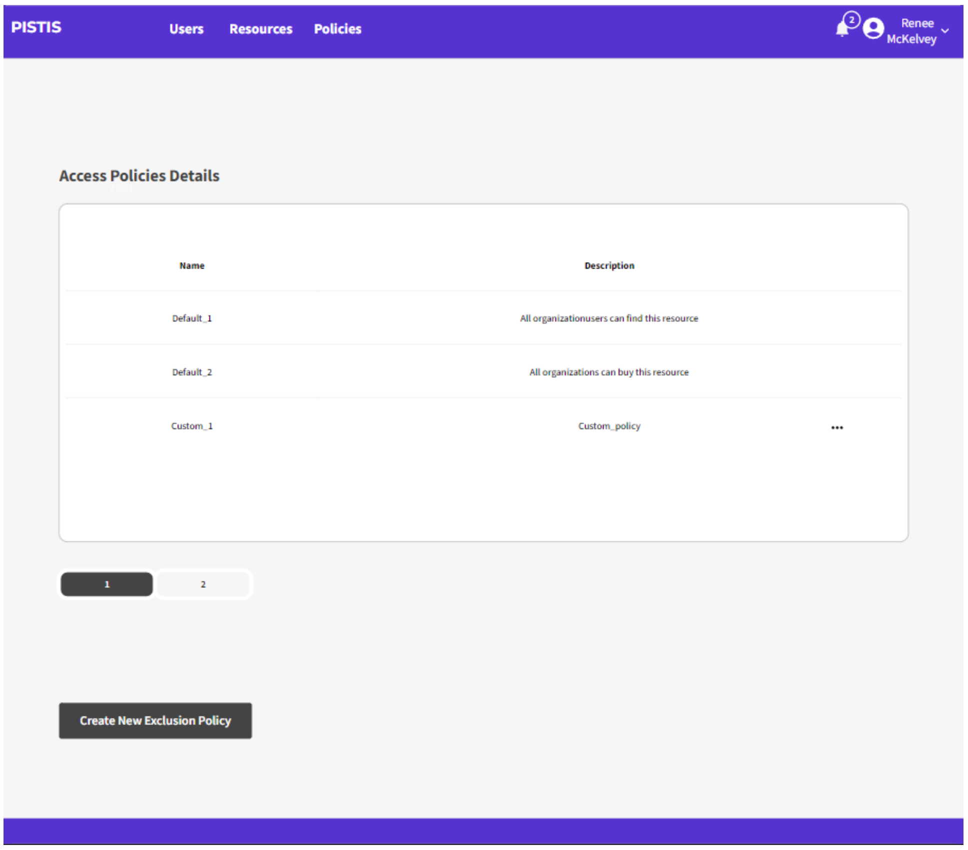
Scopes *
☐ READ ☐ EDIT

Specify users to exclude

Organization Users: User

Figure 55: Registration of a new access policy during data asset injection phase

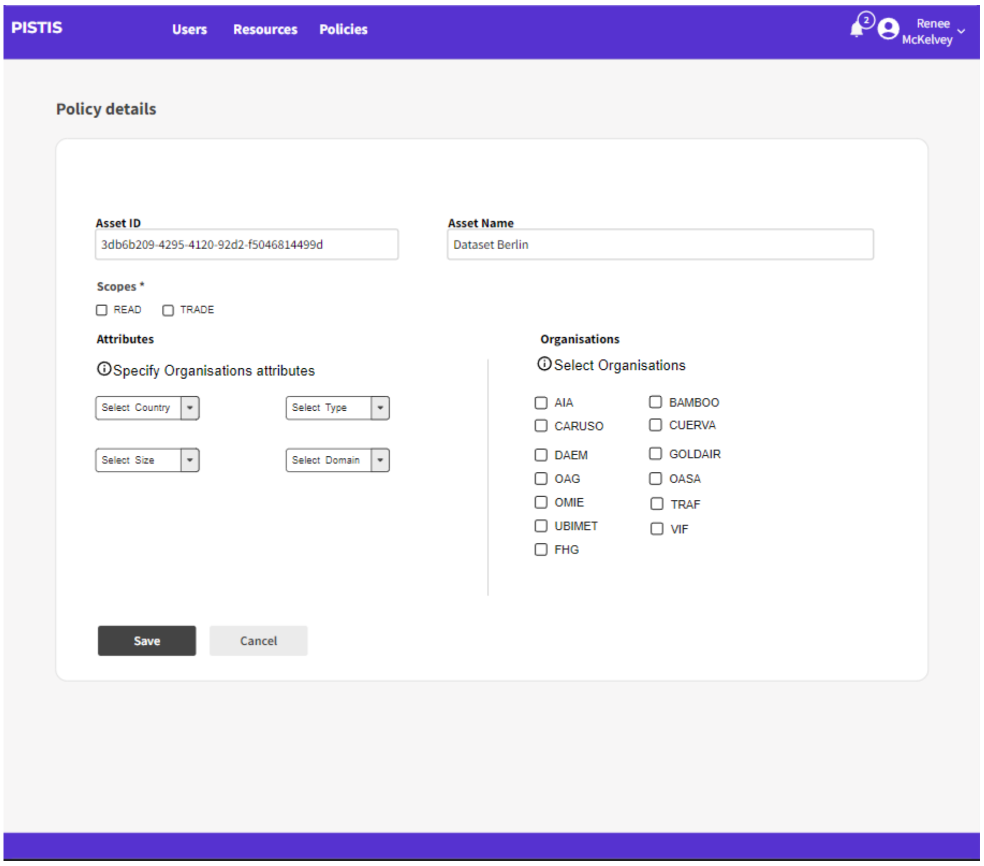
7.6.7.2 Screens related to Data Asset Publication



Access Policies Details

Name	Description
Default_1	All organizationusers can find this resource
Default_2	All organizations can buy this resource
Custom_1	Custom_policy

Figure 56: Listing of access policies during data asset publication phase



PISTIS Users Resources Policies Renee McKelvey

Policy details

Asset ID	3db6b209-4295-4120-92d2-f5046814499d
Asset Name	Dataset Berlin

Scopes *

☐ READ ☐ TRADE

Attributes

ⓘSpecify Organisations attributes

Select Country ▾ Select Type ▾

Select Size ▾ Select Domain ▾

Organisations

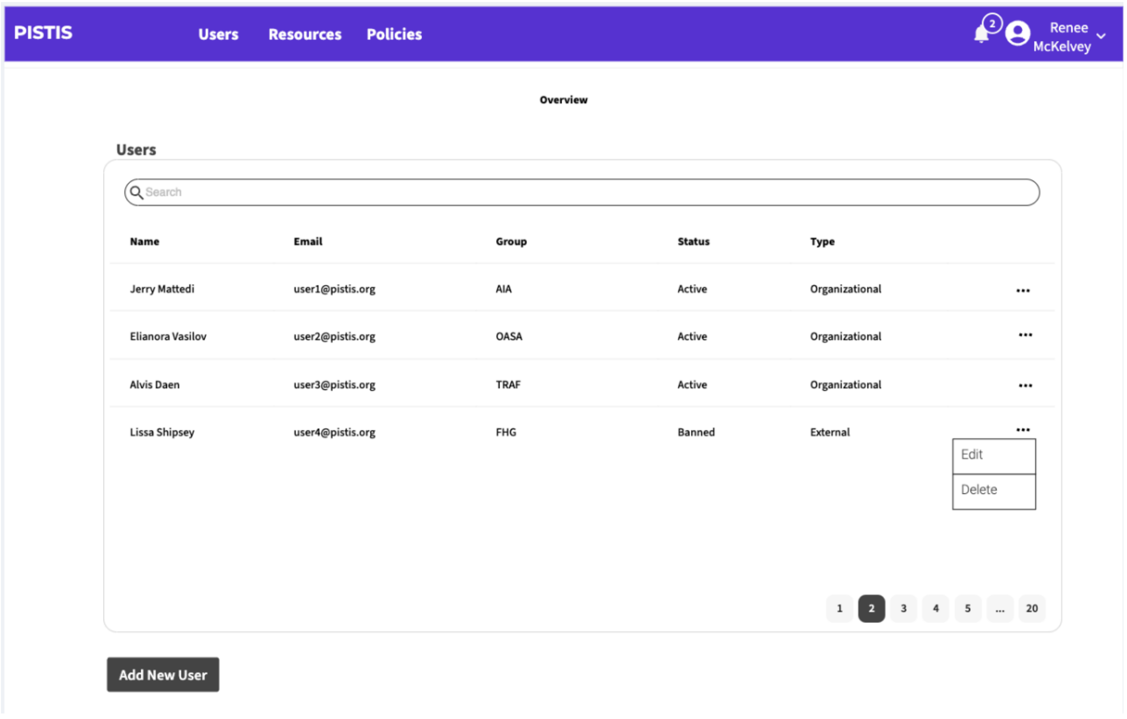
ⓘSelect Organisations

☐ AIA ☐ BAMBOO
☐ CARUSO ☐ CUERVA
☐ DAEM ☐ GOLDAIR
☐ OAG ☐ OASA
☐ OMIE ☐ TRAF
☐ UBIMET ☐ VIF
☐ FHG

Save Cancel

Figure 57: Registration of a new access policy during data asset publication phase

7.6.7.3 *Screens related to user management*



PISTIS Users Resources Policies Renee McKelvey

Overview

Users

Search

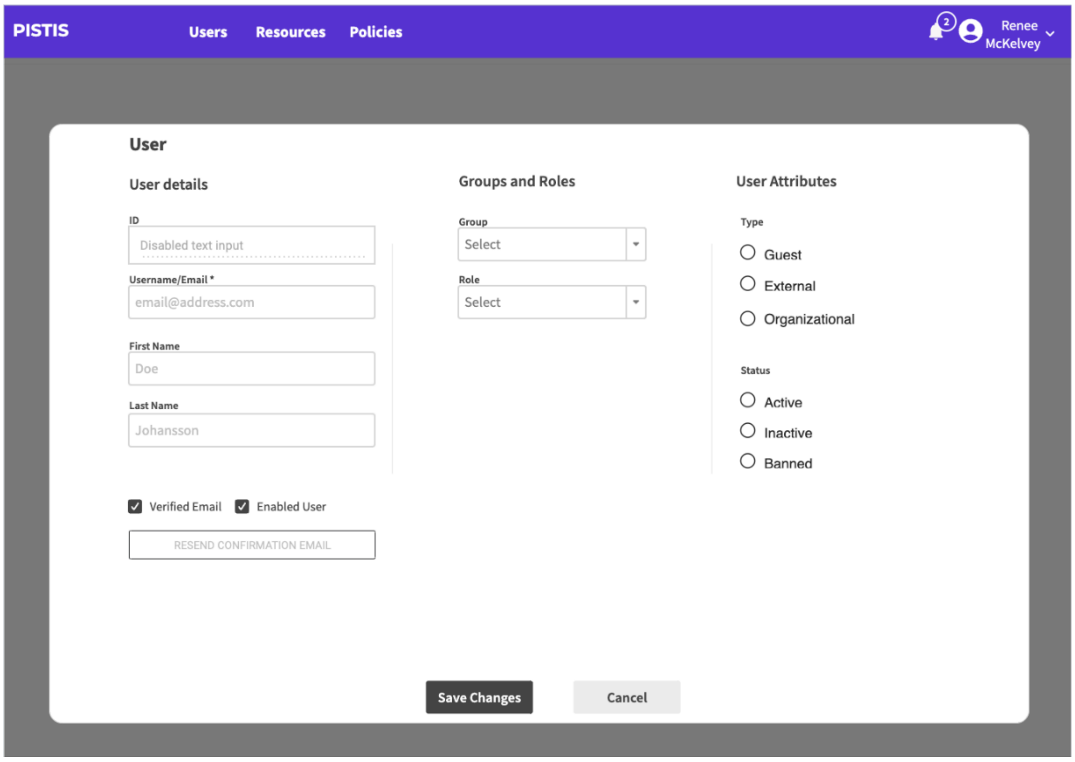
Name	Email	Group	Status	Type	
Jerry Mattedi	user1@pistis.org	AIA	Active	Organizational	•••
Elianora Vasilov	user2@pistis.org	OASA	Active	Organizational	•••
Alvis Daen	user3@pistis.org	TRAF	Active	Organizational	•••
Lissa Shipsey	user4@pistis.org	FHG	Banned	External	•••

Edit
Delete

1 2 3 4 5 ... 20

Add New User

Figure 58: Listing of PISTIS users within access policy editor



User

User details

ID: Disabled text input
Username/Email *: email@address.com
First Name: Doe
Last Name: Johansson

[✓ Verified Email] [✓ Enabled User]

RESEND CONFIRMATION EMAIL

Groups and Roles

Group: Select
Role: Select

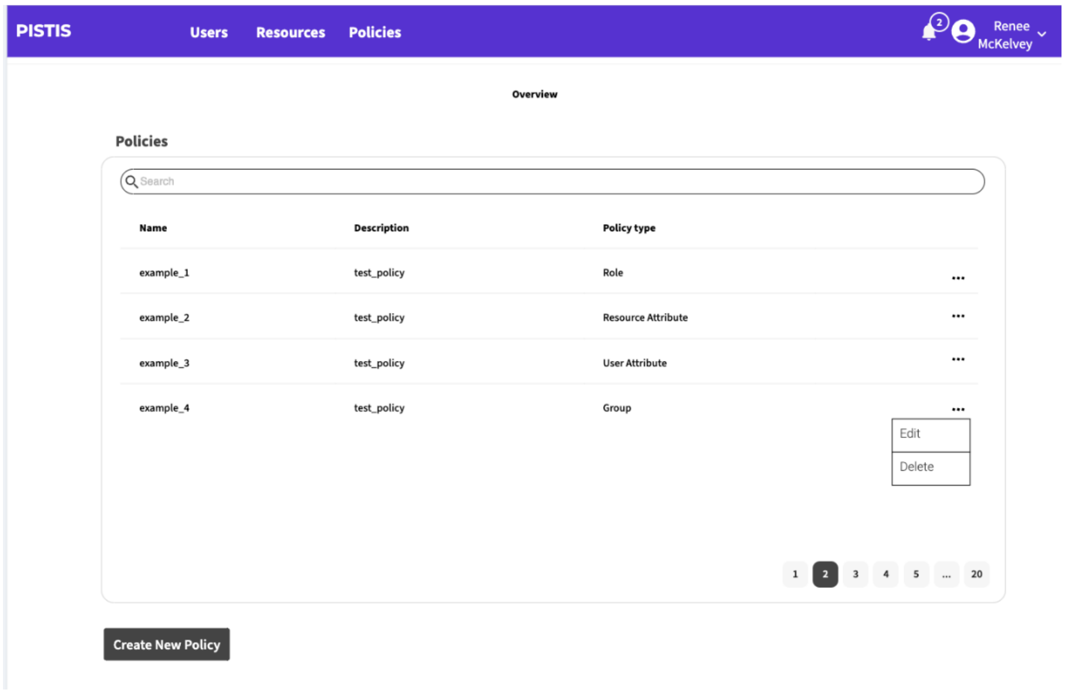
User Attributes

Type: Guest / External / Organizational

Status: Active / Inactive / Banned

Figure 59: Create (or edit) a PISTIS user

7.6.7.4 Screens related to policies management

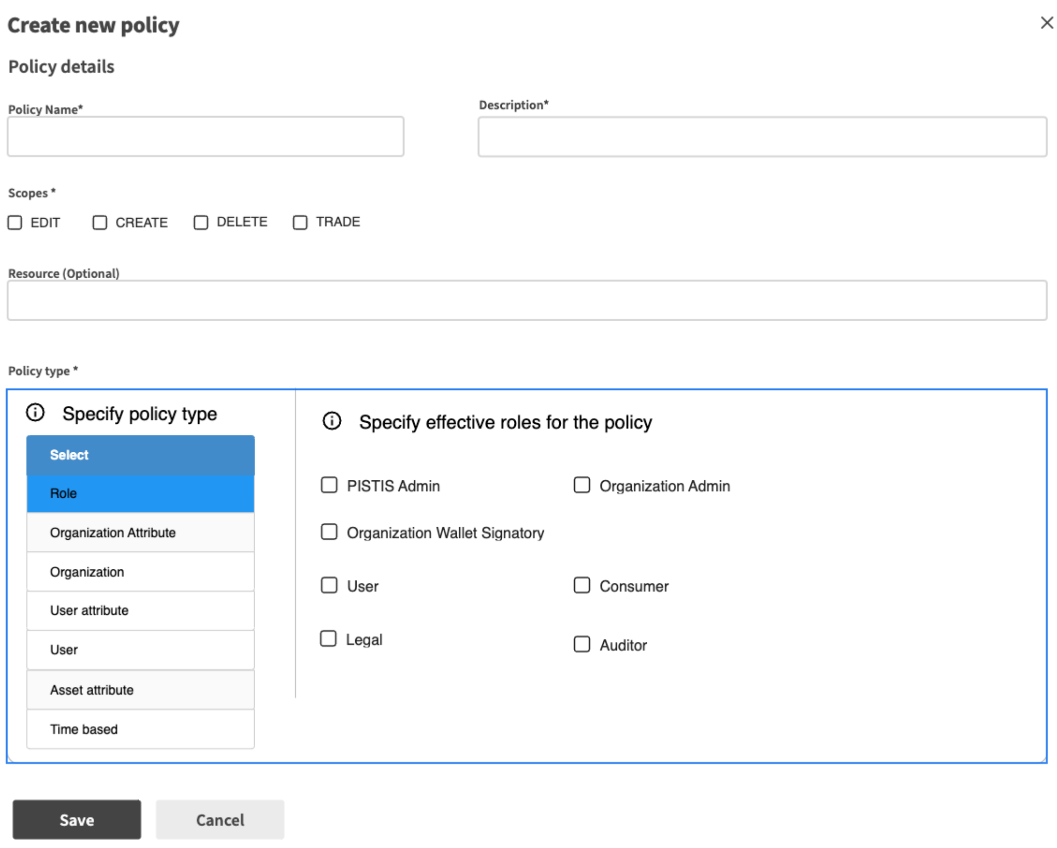


Policies

Search

Name	Description	Policy type
example_1	test_policy	Role
example_2	test_policy	Resource Attribute
example_3	test_policy	User Attribute
example_4	test_policy	Group

Figure 60: Listing of registered access policies within access policy editor


Create new policy

Policy details

Policy Name*

Description*

Scopes *

☐ EDIT ☐ CREATE ☐ DELETE ☐ TRADE

Resource (Optional)

Policy type *

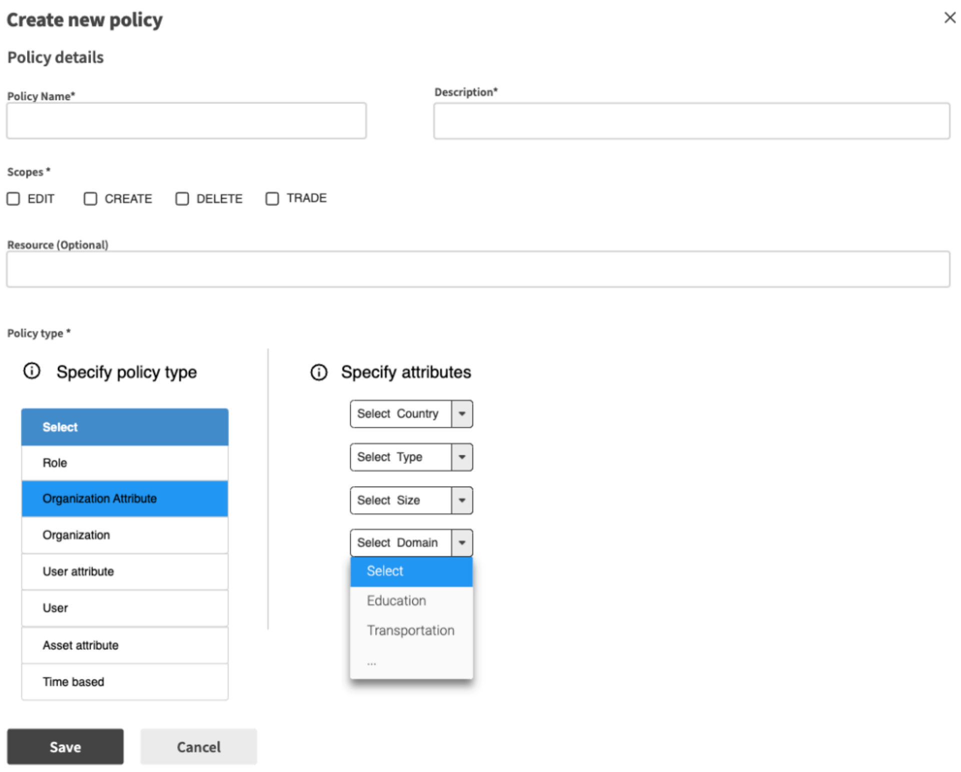
Specify policy type

Select
Role
Organization Attribute
Organization
User attribute
User
Asset attribute
Time based

Specify effective roles for the policy

☐ PISTIS Admin ☐ Organization Admin
☐ Organization Wallet Signatory
☐ User ☐ Consumer
☐ Legal ☐ Auditor

Figure 61: Registration of a new access policy within access policy editor based on user's role


Create new policy

Policy details

Policy Name*

Description*

Scopes *

☐ EDIT ☐ CREATE ☐ DELETE ☐ TRADE

Resource (Optional)

Policy type *

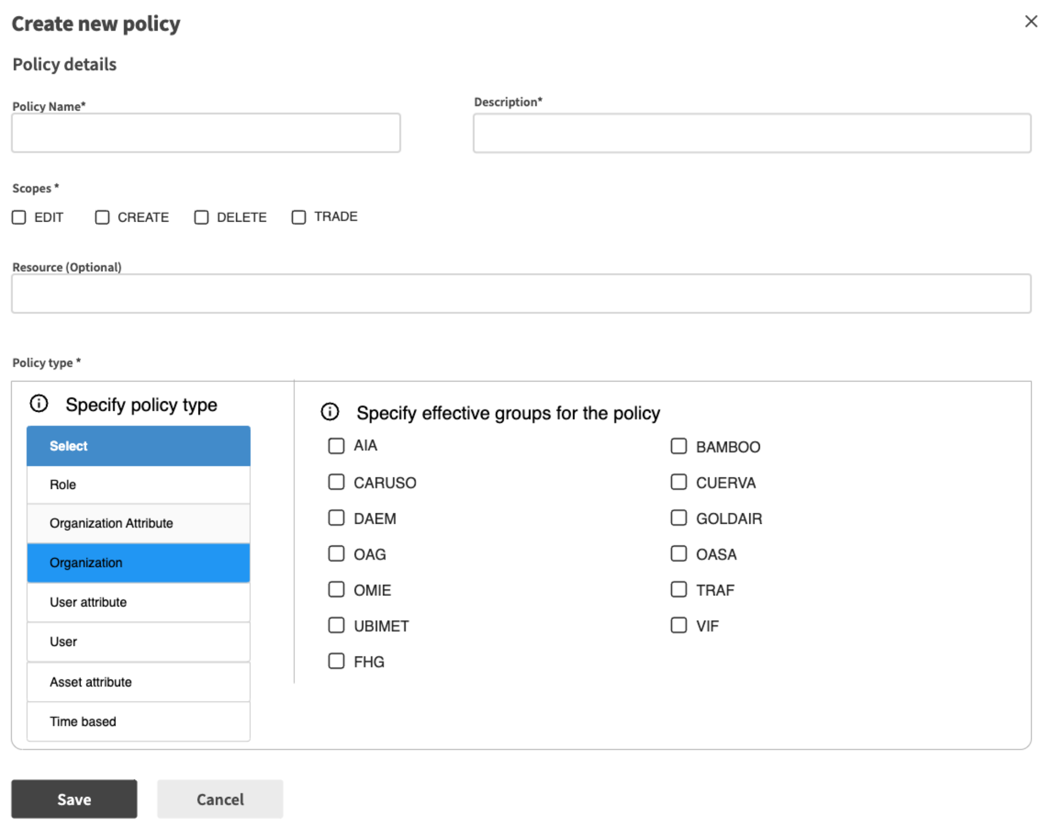
Specify policy type

Select
Role
Organization Attribute
Organization
User attribute
User
Asset attribute
Time based

Specify attributes

Select Country
Select Type
Select Size
Select Domain
Select
Education
Transportation
...

Figure 62: Registration of a new access policy within access policy editor based on user's organization attributes



Create new policy

Policy details

Policy Name*

Description*

Scopes *

☐ EDIT ☐ CREATE ☐ DELETE ☐ TRADE

Resource (Optional)

Policy type *

Specify policy type

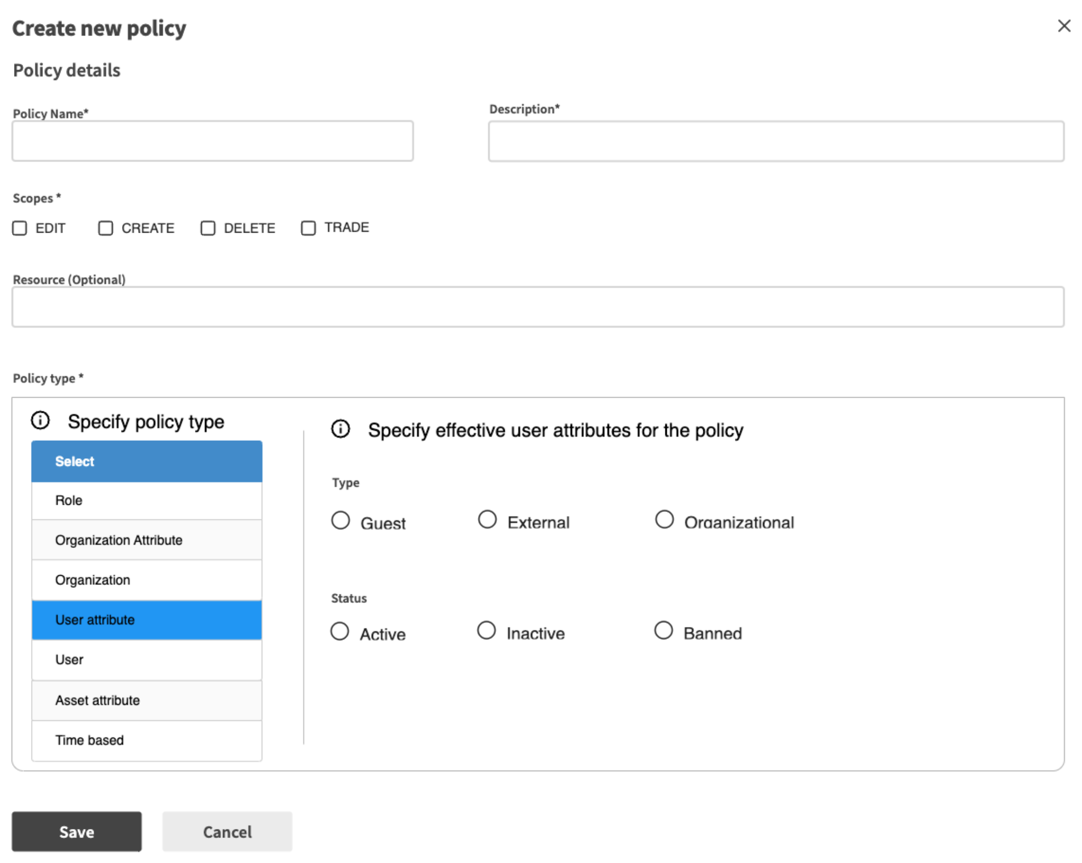
- Select
- Role
- Organization Attribute
- Organization
- User attribute
- User
- Asset attribute
- Time based

Specify effective groups for the policy

☐ AIA	☐ BAMBOO
☐ CARUSO	☐ CUERVA
☐ DAEM	☐ GOLDAIR
☐ OAG	☐ OASA
☐ OMIE	☐ TRAF
☐ UBIMET	☐ VIF
☐ FHG	

Save Cancel

Figure 63: Registration of a new access policy within access policy editor based on user's organization



Create new policy ×

Policy details

Policy Name*

Description*

Scopes *

☐ EDIT ☐ CREATE ☐ DELETE ☐ TRADE

Resource (Optional)

Policy type *

ⓘ Specify policy type

- Select
- Role
- Organization Attribute
- Organization
- [✓ User attribute]
- User
- Asset attribute
- Time based

ⓘ Specify effective user attributes for the policy

Type

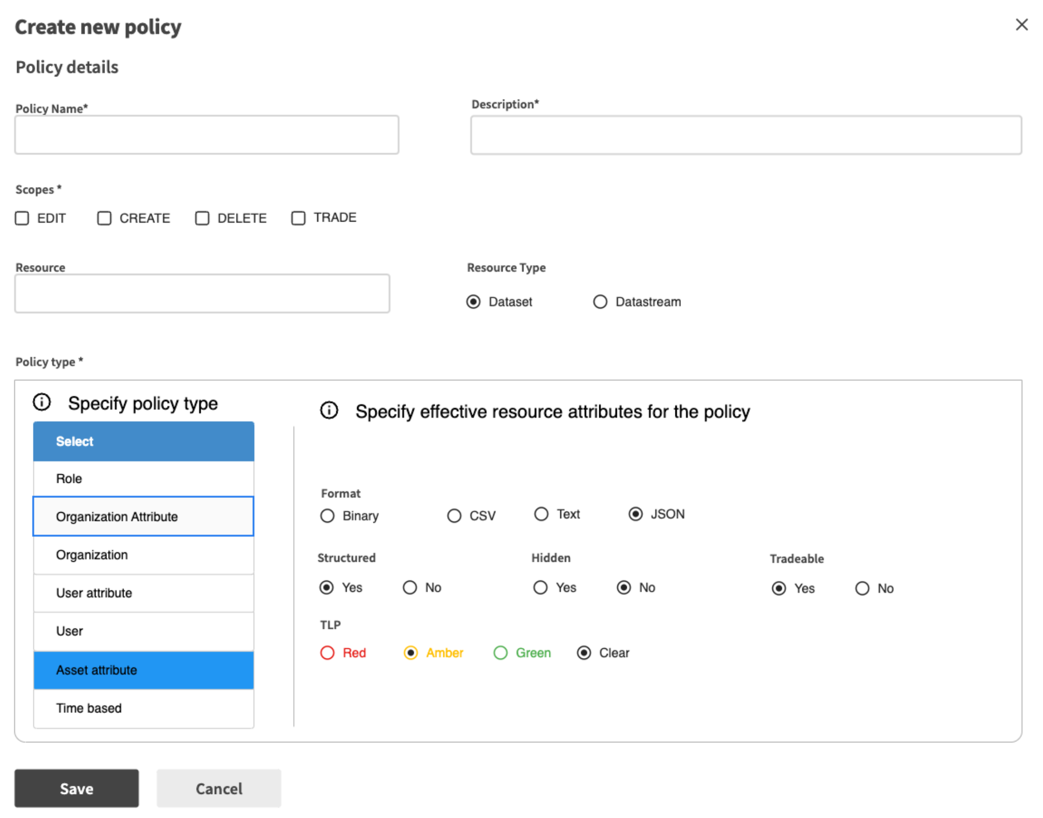
○ Guest ○ External ○ Organizational

Status

○ Active ○ Inactive ○ Banned

Save Cancel

Figure 64: Registration of a new access policy within access policy editor based on user's attributes



Create new policy ×

Policy details

Policy Name*

Description*

Scopes *

☐ EDIT ☐ CREATE ☐ DELETE ☐ TRADE

Resource

Resource Type

[✓ Dataset] ○ Datastream

Policy type *

ⓘ Specify policy type

- Select
- Role
- Organization Attribute
- Organization
- User attribute
- User
- [✓ Asset attribute]
- Time based

ⓘ Specify effective resource attributes for the policy

Format

○ Binary ○ CSV ○ Text [✓ JSON]

Structured

[✓ Yes] ○ No

Hidden

○ Yes [✓ No]

Tradeable

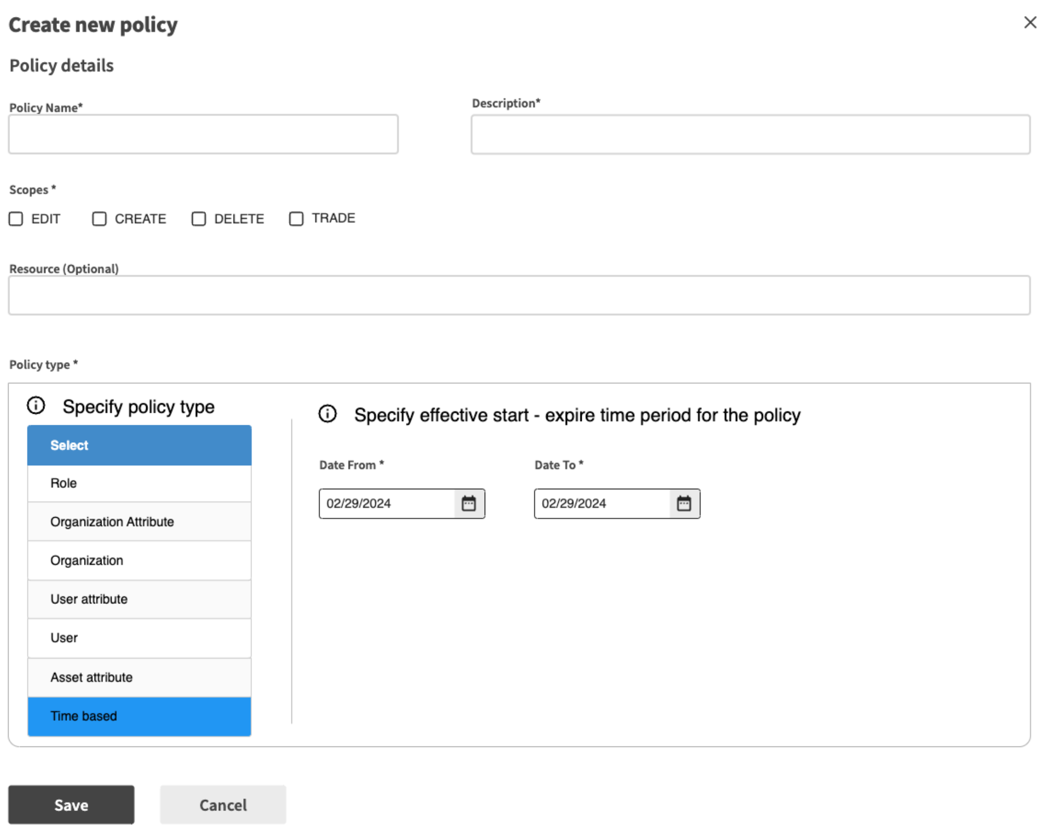
[✓ Yes] ○ No

TLP

○ Red [✓ Amber] ○ Green [✓ Clear]

Save Cancel

Figure 65: Registration of a new access policy within access policy editor based on data asset's attributes



Create new policy

Policy details

Policy Name*	
Description*	
Scopes *	☐ EDIT ☐ CREATE ☐ DELETE ☐ TRADE
Resource (Optional)	
Policy type *	Time based
Date From *	02/29/2024
Date To *	02/29/2024

Figure 66: Registration of a new access policy within access policy editor based on time period

8 CONCLUSIONS

The document presents the design and development of the Alpha release of the PISTIS components dealing with data discovery, management and protection. 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 alpha releases of the PISTIS components presented in this document and in D3.2 will drive the integration activities of the project in the next period towards realizing the delivery of the Alpha version of the PISTIS Platform (D4.2). The components will continue to evolve with updates and new features that will be encapsulated gradually in the following Beta and version 1.0 releases of the components and will be documented in the corresponding D2.3 and D2.4 deliverables.