



D3.3

Design and Implementation of the Data Governance, Quality, and Security Modules

ITI



D3.3

Design and Implementation of the Data Governance, Quality, and Security Modules

Revision **v1.0**

Work package	WP3
Task	T3.1, T3.2, T3.3, T3.4
Due date	30-09-2025
Submission date	30-09-2025
Deliverable lead	ITI
Version	v1.0
Authors	Jordi Arjona, Emili Miedes, Benjamín Navarro (ITI), Vasileios Siopidis, Nikolaos Tepelidis (CERTH), Efthymios Chondrogiannis, Efstathios Karanastasis, Anastasios Nikolakopoulos, Dionysios Diakatos (ICCS), Idoia Murua, María José López, Gorka Zarate (TEC)
Reviewers	Marcin Płóciennik, Konstantinos Kontodimas

Abstract

This document describes the components related to WP3, providing information about their design overview and, regarding their implementation, previous and final status.

Keywords

Data Governance, Data Quality, Data Security, Metadata model, Data Anonymisation, Fairness, Data Discovery, Data Ingestion, Data Storage



Document revision history

Version	Date	Description of change	Contributor(s)
v0.1	21-07-2025	Document draft	Jordi Arjona (ITI)
v0.2	29-07-2025	New content from authors	Deliverable authors
v0.3	18-09-2025	Several modifications after reviewers' comments	Jordi Arjona (ITI)
v1.0	28-09-2025	Final version	Jordi Arjona (ITI) and Santiago Cáceres (ITI)

Disclaimer

The information, documentation and figures available in this deliverable are provided by the DATAMITE project’s consortium under EC grant agreement **101092989** and do not necessarily reflect the views of the European Commission. The European Commission is not liable for any use that may be made of the information contained herein.

Copyright notice

© DATAMITE 2023-2025

Project co-funded by the European Commission in the Horizon Europe Programme

Nature of the deliverable	R
Dissemination level	
PU Public, fully open. e.g., website	✓
CL Classified information as referred to in Commission Decision 2001/844/EC	
SEN Confidential to DATAMITE project and Commission Services	



* Deliverable types:

R: document, report (excluding periodic and final reports).

DEM: demonstrator, pilot, prototype, plan designs.

DEC: websites, patent filings, press and media actions, videos, etc.

OTHER: software, technical diagrams, etc.



Table of contents

Executive summary	13
1 Introduction.....	14
1.1 Deliverable Purpose and Scope.....	14
1.2 Document Structure	14
1.3 Document Dependencies.....	15
2 Data Governance Module	16
2.1 Metadata Repository	16
2.1.1 Design Overview	16
2.1.2 Status at M18	19
2.1.3 Final Status.....	19
2.2 Data Governance Backend	22
2.2.1 Design Overview	22
2.2.2 Status at M18	23
2.2.3 Final Status.....	24
2.3 Data Catalogue, Glossary and Dictionary	25
2.3.1 Design Overview	26
2.3.2 Status at M18	27
2.3.3 Final Status.....	38
2.4 Other components.....	44
2.4.1 Data Lineage	44
2.4.2 Data Versioning.....	46
3 Data Quality Module	48
3.1 Quality Evaluator.....	48
3.1.1 Design Overview	48
3.1.2 Status at M18	50



3.1.3	Final Status.....	51
3.2	User-Defined Rules Generator.....	52
3.2.1	Design Overview	53
3.2.2	Status at M18	57
3.2.3	Final Status.....	58
3.3	KPI Library.....	60
3.3.1	Design Overview	61
3.3.2	Status at M18	62
3.3.3	Final Status.....	64
3.4	Data Quality Module frontend	65
3.4.1	Design Overview	66
3.4.2	Status at M18	67
3.4.3	Final Status.....	68
4	Data Security Module.....	72
4.1	Access Control	72
4.1.1	Design Overview	72
4.1.2	Status at M18	73
4.1.3	Final Status.....	76
4.2	Data Sovereignty.....	78
5	Data Support Tools.....	79
5.1	Data Ingestion and Storage	79
5.1.1	Design Overview	80
5.1.2	Status at M18	80
5.1.3	Final Status.....	81
5.2	Data Discovery tools	82
5.2.1	Design Overview	83
5.2.2	Status at M18	85



5.2.3	Final Status.....	86
5.3	Fairness & Data Bias	87
5.3.1	Design Overview	87
5.3.2	Status at M18	92
5.3.3	Final Status.....	93
5.4	Data Harmonisation	97
5.5	Data Anonymisation	97
5.5.1	Design Overview	98
5.5.2	Status at M18	99
5.5.3	Final Status.....	101
5.6	Data sampling recommender	102
6	Enabling Technologies.....	109
6.1	Pipeline orchestration: Mage.ai.....	109
6.2	Secret management.....	110
7	Conclusions	111
ANNEX I: APIs summary		112



List of figures

Figure 1: The Metadata Repository in the context of DATAMITE's architecture.	17
Figure 2: Metadata model	18
Figure 3: Final metadata model in DATAMITE.	21
Figure 4: Data Governance Backend in the architecture	22
Figure 5: Components the Data Governance Backend interacts with	23
Figure 6: Described frontend functionalities in the context of the architecture.	26
Figure 7: All Terms Page on the Glossary	28
Figure 8: Detail of a Term in the Glossary	28
Figure 9: Your Terms Option on the Glossary	29
Figure 10: Highlighted Terms in the Glossary.....	29
Figure 11: Vocabularies Page on the Glossary	30
Figure 12: Domains Page on the Glossary	31
Figure 13: Catalogue Page	31
Figure 14: Create a New Dataset – Type of data	32
Figure 15: Create a New Dataset – Basic Information	33
Figure 16: Create a New Dataset – Extended Information	33
Figure 17: Create a New Dataset – Permissions.....	34
Figure 18: Create a New Dataset – Confirmation.....	34
Figure 19: Detail Page of a Dataset.....	35
Figure 20: Detail Page of a Dataset - Artifacts.....	36
Figure 21: Detail Page of a Dataset - Domains	36
Figure 22: Detail Page of a Dataset - Glossary	37
Figure 23: Detail Page of a Dataset - Permissions.....	37
Figure 24: Login page	39
Figure 25: Dashboard	39



Figure 26: Dataset catalogue view.....	40
Figure 27: Data products catalogue view.....	40
Figure 28: Creation of datasets.....	41
Figure 29: Creation of datasets. Data sources - Bulk.....	42
Figure 30: Creation of datasets. Data sources - DBQuery.....	43
Figure 31: View of a dataset.....	43
Figure 32: Data Lineage and Versioning in the architecture.....	44
Figure 33: API extension to creation APIs (Data product, structuredFiles, etc.) to include lineage.....	45
Figure 34: API extension to the DBQuery API to include lineage.....	46
Figure 35: Get API returning all versions from an artifact.....	47
Figure 36: Get API returning all versions from an artifact (multiple versions exist).....	47
Figure 37: The Quality Evaluator in the architecture.....	48
Figure 38: The Quality Evaluator's operational flow.....	50
Figure 39: Data Quality Module in the DATAMITE Architecture.....	53
Figure 40: Rules creation sequence diagram.....	54
Figure 41: Rules interpretation sequence diagram.....	55
Figure 42: DATAMITE Data Quality Vocabulary schema.....	56
Figure 43: UDRG architecture.....	58
Figure 44: The KPI library in DATAMITE architecture.....	61
Figure 45: Object structure obtained from the to_dqv method.....	63
Figure 46: Example of implementation of the histogram metric.....	64
Figure 47: KPI Library internal structure.....	65
Figure 48: The Data Quality frontend in the DATAMITE architecture.....	66
Figure 49: DATAMITE Data Quality Score, Catalogue view.....	67
Figure 50: View of the Create rule interface.....	69
Figure 51: View of the Create rule interface (2).....	69
Figure 52: Example of rule defined with the UDRG.....	70



Figure 53: Example of rule defined with the UDRG specifying thresholds.	70
Figure 54: List of rules available in a dataset.....	71
Figure 55: List of rules executed in an artifact and overall score.....	71
Figure 56: The Access Control component in the context of DATAMITE's architecture	73
Figure 57: Data Access control workflow	74
Figure 58: Access Control's login page	77
Figure 59: Access Control's registration page	77
Figure 60: Data Ingestion and Storage in DATAMITE's architecture.	79
Figure 61: Data Ingestion and Storage workflows.	82
Figure 62: Data Discovery tools in DATAMITE's architecture.	83
Figure 63: Data Ingestion and Storage relation to Data Discovery Tools.....	84
Figure 64: Data Ingestion and Storage pipelines.....	84
Figure 65: Fairness & Data Bias tool in DATAMITE.	87
Figure 66: The workflow of the fairness assessment tool.....	88
Figure 67: The architecture of the fairness assessment tool.	90
Figure 68: Endpoints to create a new fairness project and filtering	91
Figure 69: Endpoints to create and run fairness queries.	92
Figure 70: Prototype UI of the Fairness and Data Bias tool	93
Figure 71: Queries tab of the fairness assessment tool	94
Figure 72: Indicative inputs to create new filtering.....	95
Figure 73: Equal opportunity metric in the Results view page.....	95
Figure 74: The Anonymisation Tool in the context of DATAMITE's architecture	98
Figure 75: Anonymisation tool workflow	99
Figure 76: SWAGGER Documentation for the Anonymisation tool	101
Figure 77: Recommender decision tree classifier visualization	103
Figure 78: Data sampling workflow	108



List of tables

Table 1: DATAMITE Security User Roles 75

Table 2: Proposed technologies for compatibility with DATAMITE..... 85

Table 3: Input data provided for training the model to recommend sampling techniques 105

Table 4: Quality module APIs..... 112

Table 5: Pipeline-related APIs..... 113

Table 6: Data Governance module APIs 116

Table 7: Data Support Tools - APIs for Fairness and Anonymisation tools..... 117



Abbreviations

AI	Artificial Intelligence
DoA	Description of Action
WP	Work Package
NLP	Natural Language Processing
JSON	JavaScript Object Notation
CSV	Comma Separated Values
RBAC	Role-Based Access Control
DQV	Data Quality Vocabulary
DDQV	DATAMITE Data Quality Vocabulary
UDRG	User Defined Rules Generator
QE	Quality Evaluator
NLP	Natural Language Processing





Executive summary

D3.2 provides a detailed overview of the status of the different components related to WP3. The goal of WP3 was to develop those tools that users need to improve the management of data within the organisation. These tools span aspects like the governance of data, being able to visualise the quality of data, or configure the access control to datasets; configure different types of data ingestion or discovery, anonymising data or detecting and mitigating fairness problems, among others. Having a solid set of tools to improve the management within the company will enable the organisation to share interoperable and high-quality data, leveraging the data sharing tools produced in WP2

Thus, this deliverable provides information on the design overview, intermediate status at M18 and final status for the components comprised in the Data Governance, Data Quality and Data Security Module, as well as in the Data Support tools. Similarly, it includes a description of how DATAMITE brings in some additional technologies as enablers, as could be the case for pipeline management in the context of data discovery tools and data ingestion, or secret vaults for storing passwords to remote databases.



1 Introduction

The DATAMITE project is funded by the European Commission as part of the Horizon Europe programme and coordinated by the ITI - Technological Institute of Informatics. DATAMITE empowers European companies by delivering a modular, open-source and multi-domain Framework to improve DATA Monetising, Interoperability, Trading and Exchange, in the form of software modules, training, and business materials.

DATAMITE unleashes the monetisation potential at two levels. At internal level, users will have tools to improve quality management of their data, the adherence to FAIR principles, and will be able to upskill on technical and business aspects thanks to the multiple open-source training materials the project will generate. Better data and personnel with higher skills will help companies to improve their data culture and shift to more data-driven approaches.

At external level, keeping organisations in control of their data will provide new sources of revenue and interaction with other stakeholders. The architecture envisioned for DATAMITE enables DIHs sandboxing, becoming a potential instructor on their onboarding of SMEs and low-tech SMEs into the data economy. Together, DATAMITE's solutions will act as a catalyst to boost data monetisation in the European productive fabric.

1.1 Deliverable Purpose and Scope

Specifically, the Grant Agreement states the following regarding this Deliverable: describes the design of the modules, the integrated building blocks, and their implementation. This includes the improvements proposed in terms of data ingestion and storage, and associated supporting tools.

Hence, the purpose of this document is to present the status of the different components related to WP3. For each of these components, this document presents its current design, the status regarding implementation, and the next steps to be undertaken.

1.2 Document Structure

This deliverable is broken down into the following sections:

- **Executive Summary:** A summary of the contents and main findings are provided.



- **Section 1 Introduction** provides the deliverable general context, dependencies, and structure.
- **Section 2 Data Governance Module, Section 3 Data Quality Module, Section 4 Data Security Module and Section 5 Data Support Tools** provide a status overview and the next steps of the components comprised in each module.
- **Section 6 Enabling Technologies** describes those technologies that are being used transversally to different modules as enablers of different functionalities.
- **Section 7 Conclusions** closes the document.

1.3 Document Dependencies

This document is part of an iteration of living deliverables. The first version was delivered in M12, describing the initial design of the different components belonging to modules related to WP3. This second iteration presented the evolution of such components at M18, jointly with the planned steps for their evolution. This is the last document in this series, presenting the final version of the different components that have been developed under the umbrella of WP3.

2 Data Governance Module

This section presents the final status of the Data Governance Module components, namely, the Data Governance Backend, Metadata Repository and several frontend functionalities. It presents the progress performed since M18, when the previous version of this deliverable was delivered. As a result, their design, previous and final status are presented.

2.1 Metadata Repository

Apache Atlas was selected as the metadata repository for DATAMITE, as was explained in deliverable *D3.1 Design and Implementation of the Data Governance, Quality, and Security Modules*. The main reason for this selection is that it was considered to be slightly more flexible and open than other possible alternatives (e.g., OpenMetadata or LinkedIn Datahub), though more of a building block than an end-user service. Besides its functionalities, the most important aspect of the metadata repository is the metadata model we devised and deployed in Apache Atlas. It allowed us to create a fully customised and extensible metadata model for the framework. The metadata repository is isolated from the rest of the components in the architecture and interfaces solely with the Data Governance Backend, as shown in Figure 1.

2.1.1 Design Overview

The main development in the Metadata Repository is the design and deployment of the metadata model. The initial version of this metadata model is shown in Figure 2. The goal of the model is to accurately represent as entities those objects (e.g., datasets, artifacts, data products) that will be displayed in the catalogue. Entities contain three types of information:

- Metadata that is captured or generated automatically. First, the object's metadata (e.g., a table) that already exists (e.g., column names, owner, descriptions) in the data source. This metadata is extracted from the data source without being altered by users. It can be separated into two categories: technical aspects (e.g., structure, creation date, creation user) and business aspects (e.g., descriptions, data origin). Second, metadata that is generated when ingesting the data, such as creation date, dataset owner, etc.

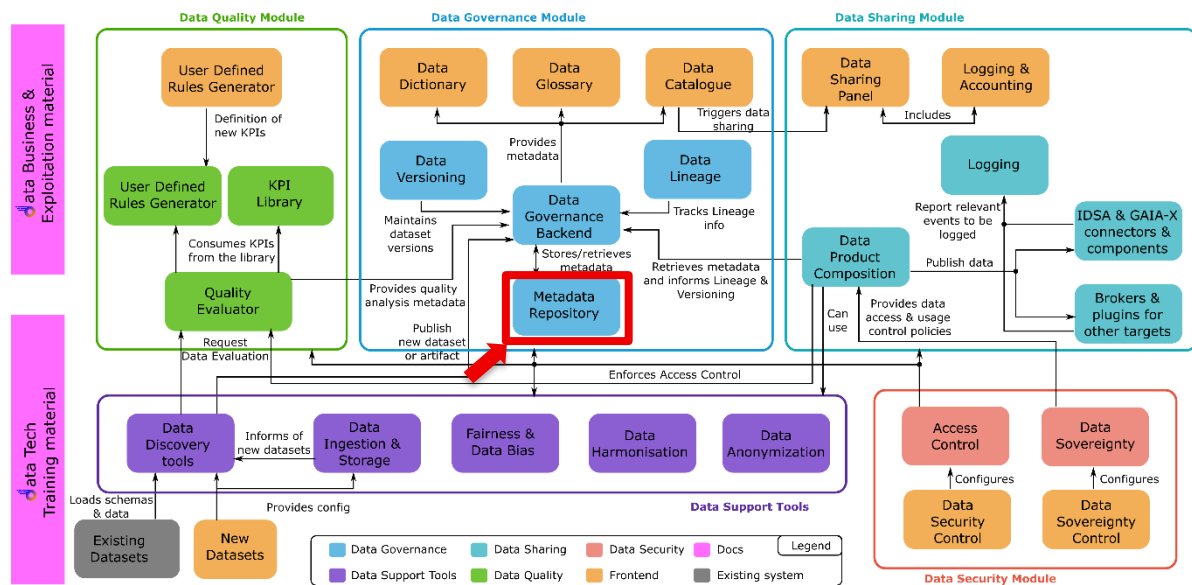


Figure 1: The Metadata Repository in the context of DATAMITE's architecture.

- Quality metadata. These metadata are generated during the profiling process of a data artifact. It is stored as Quality Measurement entities, following the Data Quality Vocabulary (DQV) specification¹, and provide information related to inherent quality indicators (e.g., completeness, uniqueness, statistics) or user-defined indicators.
- Metadata provided by the users. These metadata are created by the users through the catalogue interfaces to enhance their understandability and context, e.g., entity names, entity descriptions, glossary terms, and classifications.

Glossary terms belong to specific vocabularies that users can create in the Data Glossary. These terms can be linked to entities in the catalogue to provide context-univocal information to data users. This metadata aims to enhance interoperability and adherence to the FAIR principles² by improving findability or accessibility, especially for non-technical users.

¹ [Data on the Web Best Practices: Data Quality Vocabulary \(2016\)](#).

² Mark D. Wilkinson et al. *The FAIR Guiding Principles for scientific data management and stewardship*. Scientific Data 2016.

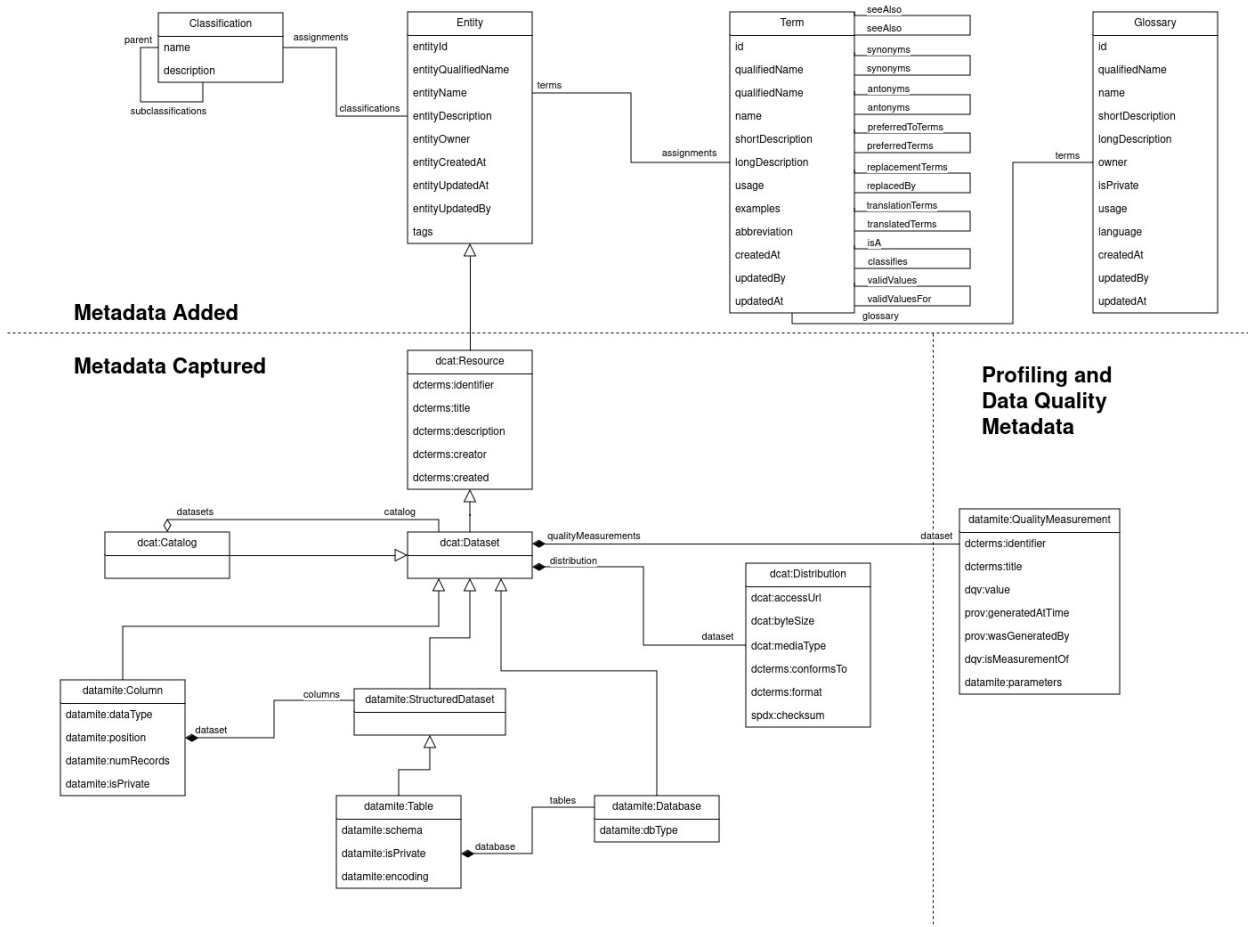


Figure 2: Metadata model

This model is based on DCAT³, which is probably the most popular specification for data catalogues and is thought to facilitate its interoperability. However, DCAT is not detailed enough for DATAMITE, as its main goal is to facilitate interoperability between data catalogues published on the Web, not needing to delve deep into the internal structure or elements of the dataset. Thus,

³ Data Catalog Vocabulary (DCAT) - Version 2 (2020)



potential extensions are being studied by the research team, such as the ones shown in the Figure 2, so complex data assets can be considered with a finer grain.

The richness of the metadata model is key to providing solid foundations for other components, such as the Data Catalogue or the Data Glossary, which will be the main sources of interaction with the users and, hence, of metadata enrichment for the data assets. Similarly, a detailed set of metadata is needed for the additional development of data products, which can be created by taking a dataset from the catalogue or extracting subsets of existing datasets or combining them into a data product.

2.1.2 Status at M18

The schema deployed in the Apache Atlas development environment is depicted in Figure 3. The evolution of versions in previous deliverables is mostly obtained by adding databases to the schema and aligning them with DCAT. The latter also influenced how entities were usually described in the project. For instance, previously, the top level was referred to as a dataset, and it was composed of artifacts. Now, according to the DCAT notation, formally, a dataset should be a catalogue, and its elements would be datasets.

The metadata model's development status is mature but not yet complete. Pending tasks at M18 were, among others, to adapt to the developments coming from WP2 regarding data products or data sovereignty.

2.1.3 Final Status

The metadata model has reached its final state over the last months. It has included several new entities destined to give support to the new developments of the project. The main additions to the metadata model during the last period have been:

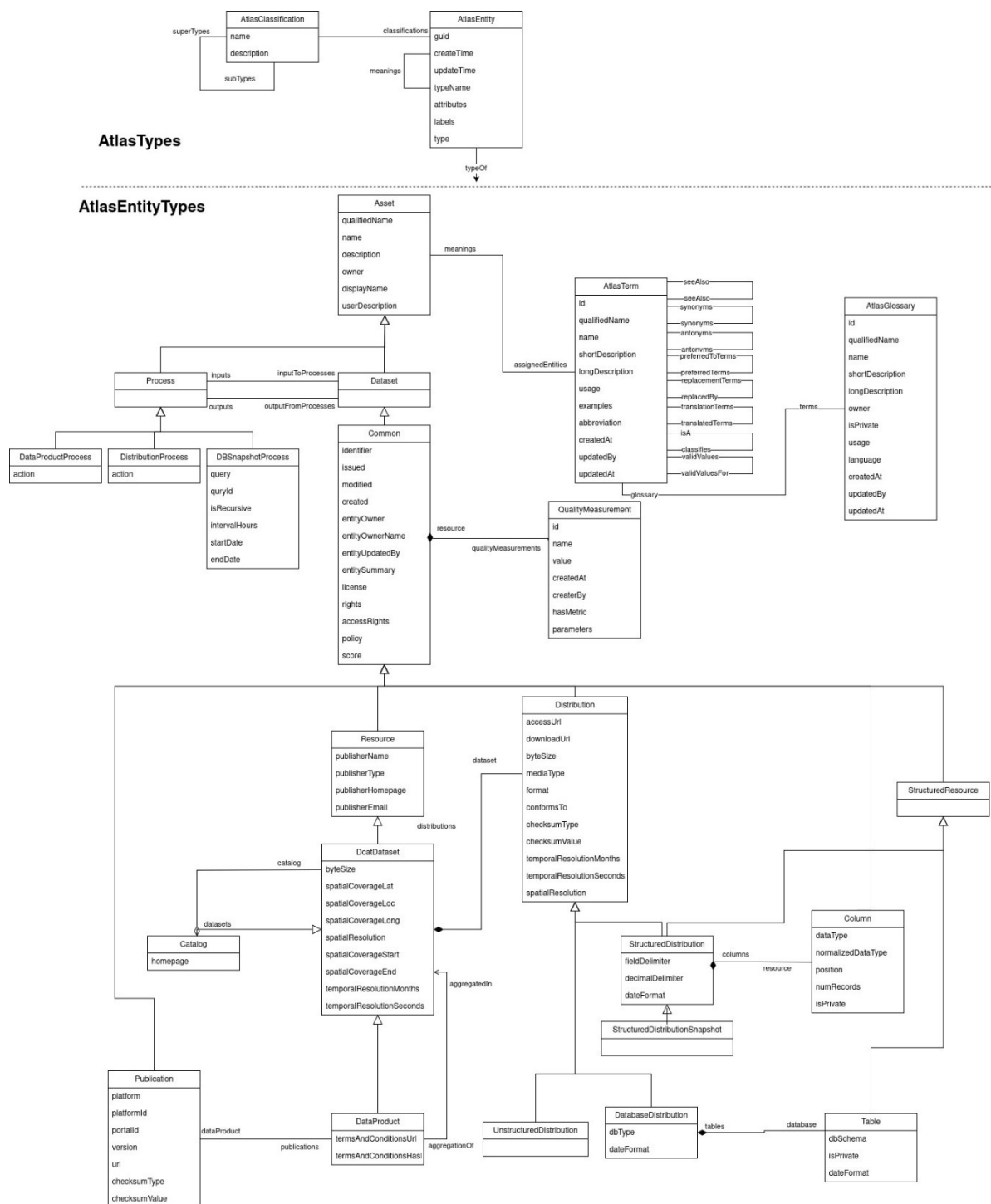
- Substantial refactoring of existing entities to increase the alignment to standards, concretely DCAT. This was related to the interpretation made of what an artifact was and how datasets are composed of artifacts. This affected the different relations of composition between entities and required the definition of some new ones.
- Addition of Data Product related entities. These entities are aligned with the approach followed for datasets, that is, they consider a parent level, the data product, which can



include several data artifacts. Data products include a series of specific fields more oriented to their eventual sharing, e.g., spatial or temporal coverage, or licenses, among others.

- Addition of Publication related entities. Publications are needed to store the information related to the different data products shared by the data provider through the different connectors or brokers. It includes fields such as the ID of the publication, the data space, portal or initiative where it was published, the corresponding URL, or the internal URL pointing to the data shared, among others.
- Addition of DB Queries related entities. Include specific information related to the queries that users can perform on already identified databases to extract specific data. Examples would be the query itself or the databases involved. The data resulting from a query will be, in principle, used as a new artifact for an existing dataset or to create a new one.
- Addition of entities for unstructured data. A common entity has been defined to describe non-structured data. In principle, this entity contains generic information common to any kind of object, such as name, type, format, etc. These fields will provide the generic information for audio, images, video or others, while specific indicators related to its quality (bitrate, density, etc) may be computed and stored as quality metadata.
- Enabling the relations between glossary terms. Allows for defining and storing different types of relations between terms in a glossary, such as synonymy, antonymy, similarity, etc.
- Enabling data lineage. New entities have been included to store the lineage of the different artifacts registered in the metadata repository, capturing the origins and actions performed on them.

These new additions cover the different demands that have been identified in the pilots as well as by other components (e.g., the Data Product Composition). The integration with the Access Control Module has been considered, although it is delegated to the Data Governance Backend. Figure 2 and Figure 3 can be compared to see the evolution of the metadata model.



2.2 Data Governance Backend

Figure 4 shows the Data Governance Backend in the context of the architecture. The Data Governance Module receives the metadata produced by the Data Discovery tools and the Data Quality Module. The Data Governance Backend aims to act as an interface to the Metadata Repository, abstracting the metadata model from it. Moreover, the Data Governance Backend provides the API to interact with the metadata repository. This design was made following principles of flexibility and modularity to be able to replace this component in the future, if required, with minimum impact on the rest of the framework.

2.2.1 Design Overview

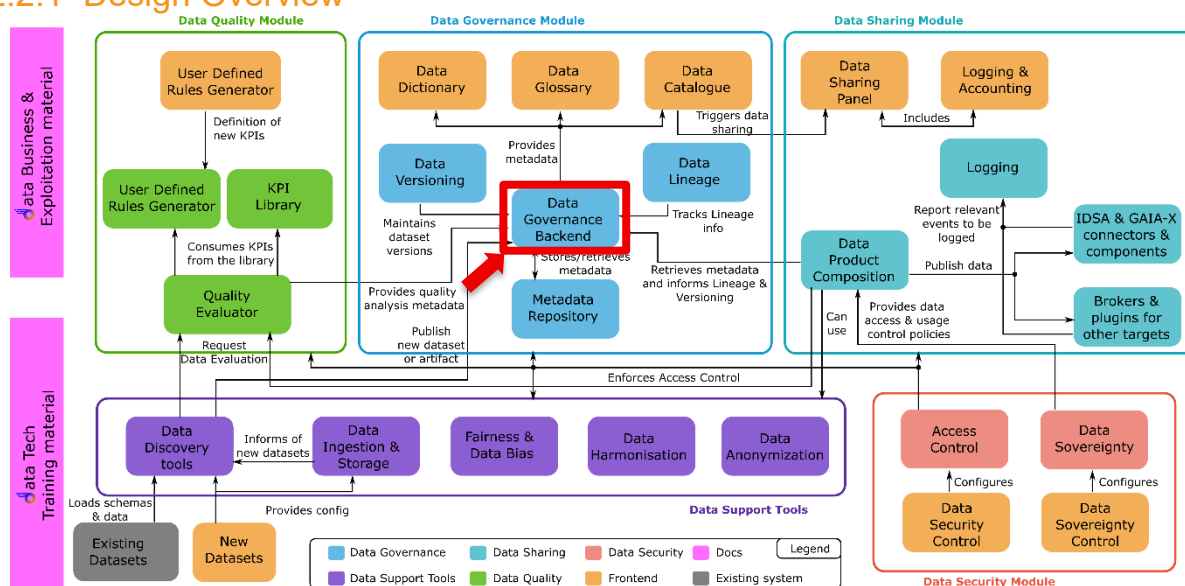


Figure 4: Data Governance Backend in the architecture

The goal of the Data Governance Backend is to act as an interface between the framework and the Metadata Repository. Hence, it will be the source and sink of metadata information for the rest of the components of the framework. As can be seen in Figure 5, the Data Governance Backend is expected to receive metadata, at least, from the connectors from the Data Discovery tools component, which will push structural metadata from the different datasets or artifacts that are discovered, either stored locally or remotely (see the Data Ingestion & Storage, the Data Discovery tools components, Sections 5.1 and 5.1) or the Data Product Composition, among

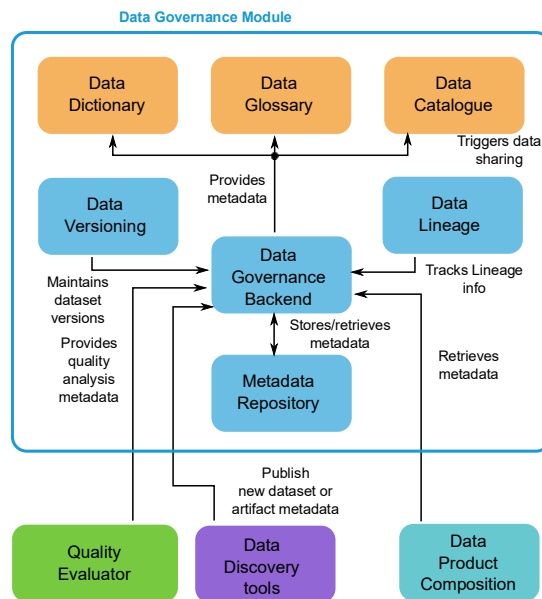


Figure 5: Components the Data Governance Backend interacts with

others. It will also receive quality metadata information computed by the Quality Evaluator, although the associated pipeline is still being defined, and the metadata may be provided directly by an intermediate component. It also receives information related to data versioning lineage.

On the other hand, the Data Governance Backend will provide information to the Data Catalogue, Glossary and Dictionary (note that there is a Dictionary per data artifact), which are the main user interfaces in the framework, and also to the Data Product Composition component, which will be in charge of creating data products from datasets in the catalogue.

The following subsections provide more information on the APIs provided and the status of the Data Governance Backend.

2.2.2 Status at M18

The Data Governance Backend provides an API allowing the abstraction of the metadata model deployed in the metadata repository (see Section 2.1 and Figure 2). The API allows different operations related to:

- The creation, management, and deletion of datasets.



- The creation, management and deletion of data artifacts: File, Database, Table or Column type.
- The creation, management and deletion of vocabularies and their associated Terms.
- The linking of terms from a vocabulary in the Glossary to other entities, such as columns, tables, databases, etc.
- Search operations.

A detailed description of the implemented API and the endpoints in the previous categories can be found in the [DATAMITE's Eclipse Repository](#)⁴. Similarly, a development environment for testing and integration has been deployed at ITI's computing infrastructure, jointly with other components such as the metadata repository and the discovery connector tools.

2.2.3 Final Status

The Data Governance backend has grown in this period to provide a series of new APIs covering all the aspects of the metadata model. Namely, the most relevant groups of APIs added during this period would be the following:

- Data products API: Provides the APIs to create, retrieve, update or delete data product entities in the metadata repository, as well as other operations related to them. These APIs are mostly consumed by the Data Product Composition component, but also by the frontend, or by the data space connectors or brokers during the process of sharing data.
- Publications API: It offers APIs to store the metadata related to those data publications made on the different available targets (data spaces, AI-on-Demand platform, etc) and to recover or update them when needed. These APIs are mostly consumed by the frontend, data space connectors and brokers.

⁴ <https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-governance/data-governance-backend>



- DB Query API: Implied modifying some existing APIs as well as adding new ones to store this new kind of entity related to this new kind of ingestion tool. It is used by the data ingestion tools.
- Updates regarding the quality regarding the score/final score
- Relations between terms: New APIs have been added to enable the creation of different types of relationships between glossary terms, taking advantage of the basis provided by Apache Atlas.
- Unstructured data API: New APIs have been provided for the management of metadata related to unstructured data artifacts, such as audio, video or images.
- Refactoring of existing governance APIs: Different APIs had to be updated to adapt to the evolution of the metadata model.
- Integration with Access Control

Similarly, the module has integrated with the access control module, validating that users consuming the different APIs are allowed to consume them. In general, all the tasks that were planned ahead have been covered.

2.3 Data Catalogue, Glossary and Dictionary

The goal of the frontend user interfaces and experiences for the various components of the DATAMITE platform is to ensure that users can effectively manage, govern, and monetise their data while adhering to best practices and regulatory requirements. The Data Catalogue, Glossary and Dictionary are some of the main tools for the user regarding tasks such as uploading datasets, inspecting their quality or working with them. This section refers to the boxes marked in red in Figure 6, which involve the main Dashboard/Homepage, the Catalogue (where the datasets are displayed), the Glossary (Terms and Vocabularies) and the Domains, jointly with the Dictionary, which provides the basic information about data quality.

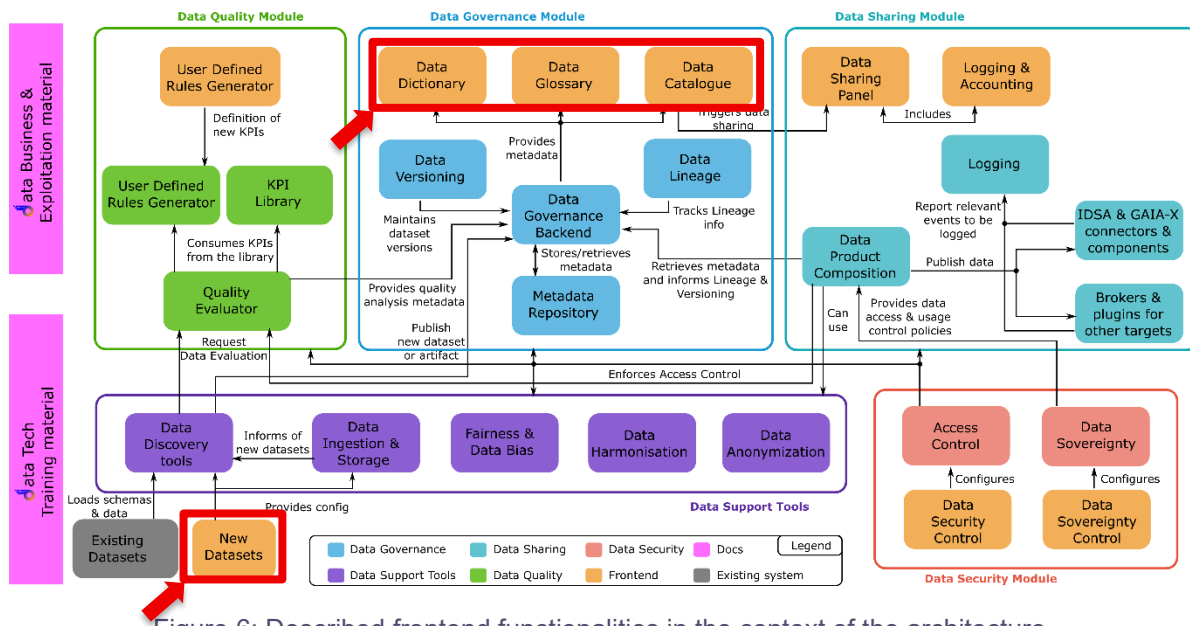


Figure 6: Described frontend functionalities in the context of the architecture.

2.3.1 Design Overview

The components involved in this section provide part of the main interface structure of the framework. The dashboard will be the landing page for the user, providing information about its datasets, data products, or publications, as well as providing access to the Glossary, ingestion mechanisms or other tools.

The Data Catalogue allows the user to have quick access to its datasets, with easy access to some basic information such as the creation date, quality score, description or linked terms. When the user accesses a dataset, she gets access to the artifacts composing it, extended information about their columns, a brief quality overview, extended description, access to domains, permissions or the ability to enrich the dataset with glossary terms, among others. Other functionalities, like adding new artifacts or defining user quality rules or visualising their scores, can also be accessed from here.

From the data artifacts detail, the users can access the Data Dictionary for the different columns, obtaining quality information about their quality, and seeing the results for the different KPIs included in the KPI Library.



Finally, the glossary will provide the user access to the different vocabularies she has defined, as well as to the different terms within. The user, according to their permissions, will also be able to edit or create new terms, as well as see where those have been used to enrich existing entities in the catalogue.

Each screen has an initial design phase based on wireframes, a second richer phase based on the use of Figma, and, afterwards, is developed and brought into a production environment.

2.3.2 Status at M18

Regarding the design, the wireframes were mostly completed for the main pages by month 18, and the [UI Design](#)⁵ has started for the Dashboard, Data Catalogue, Data Glossary and Data Dictionary pages. The Dashboard refers to the homepage, the Data Catalogue includes all the information of the datasets and data products, and the Data Glossary and Dictionary pages initially referred to the Govern component option (now obsolete), which linked to the Glossary, which includes the vocabularies and terms, and Domains pages. Regarding the [frontend development](#)⁶, the main structure of the dashboard, landing page and the Data Catalogue components were almost completed.

Finally, a back office for the admin was developed as well to manage content on the DATAMITE platform, like the FAQs, translations, banners and administration options to manage which DATAMITE modules are activated or not. The following figures show part of the work done or that was being carried out at this stage.

First, Figure 7 and Figure 8 provide an overview of the Data Glossary interface. Based on the permissions of the user profile, the user can edit, share, download or delete it. On the left side menu, the user can also see the “your terms”, as shown in Figure 9, which are the ones created by the user. On the other hand, Figure 10 shows the highlighted terms option.

⁵ <https://tinyurl.com/datamite-ui>

⁶ <https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/frontend>

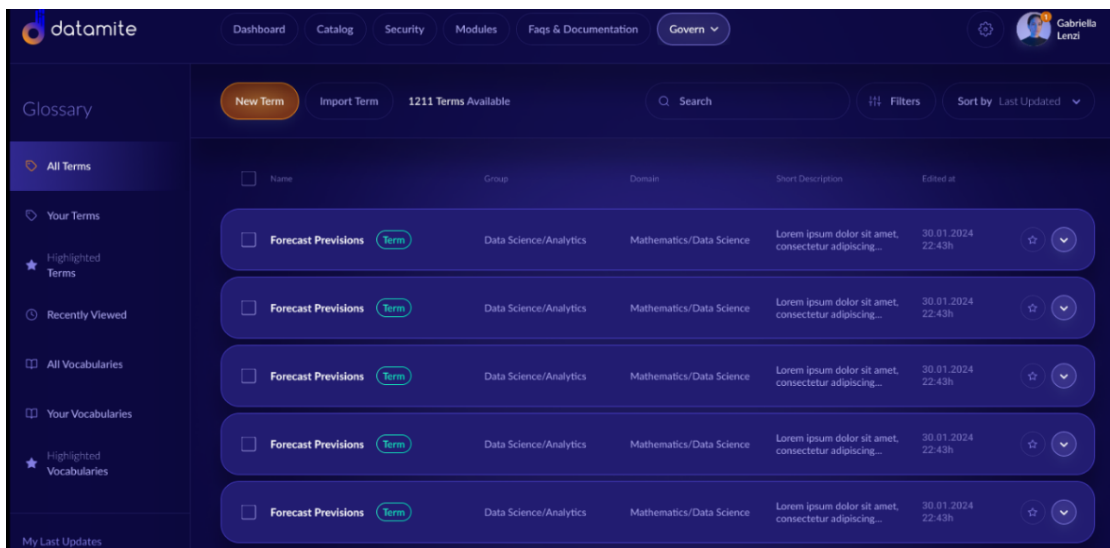


Figure 7: All Terms Page on the Glossary

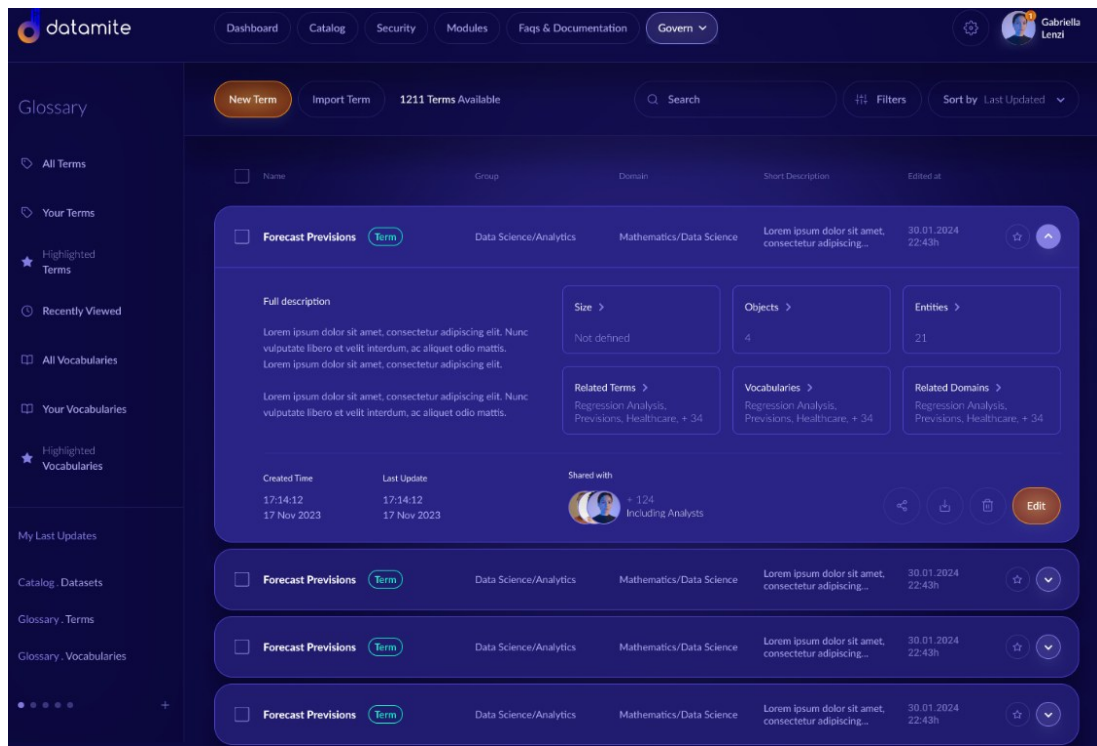


Figure 8: Detail of a Term in the Glossary

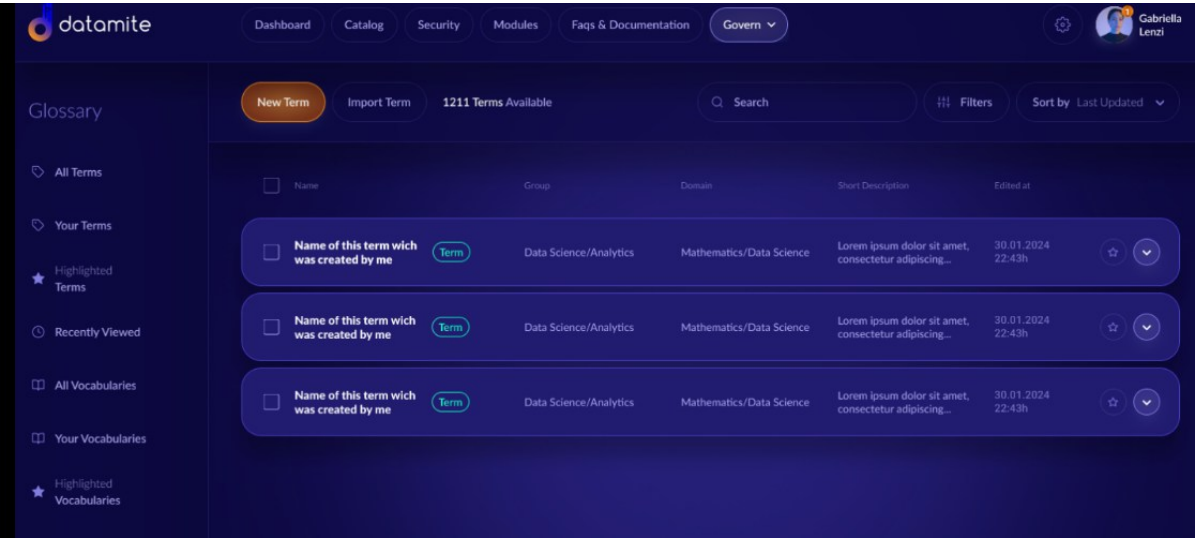


Figure 9: Your Terms Option on the Glossary

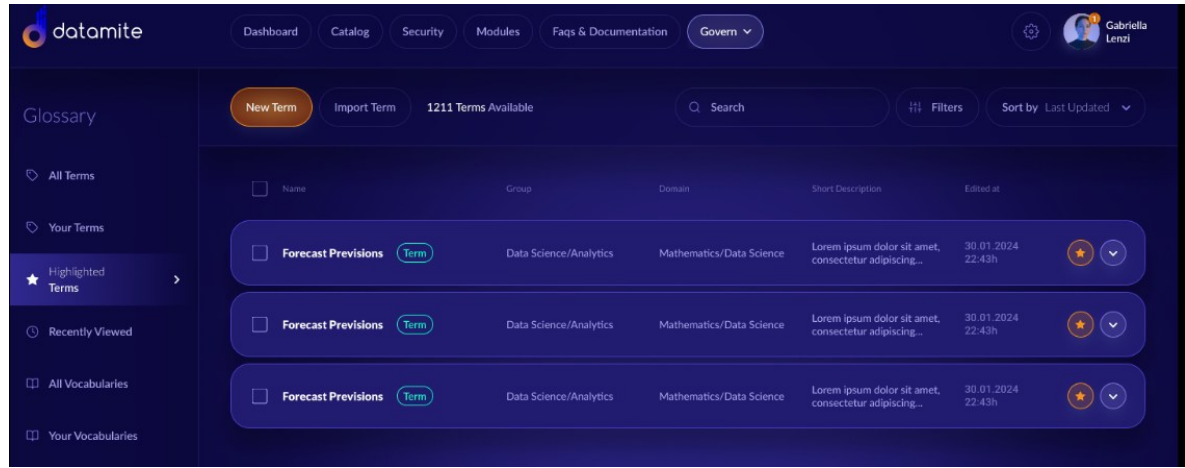


Figure 10: Highlighted Terms in the Glossary

Regarding the available vocabularies, they are displayed by using listing cards, as can be seen in Figure 11.

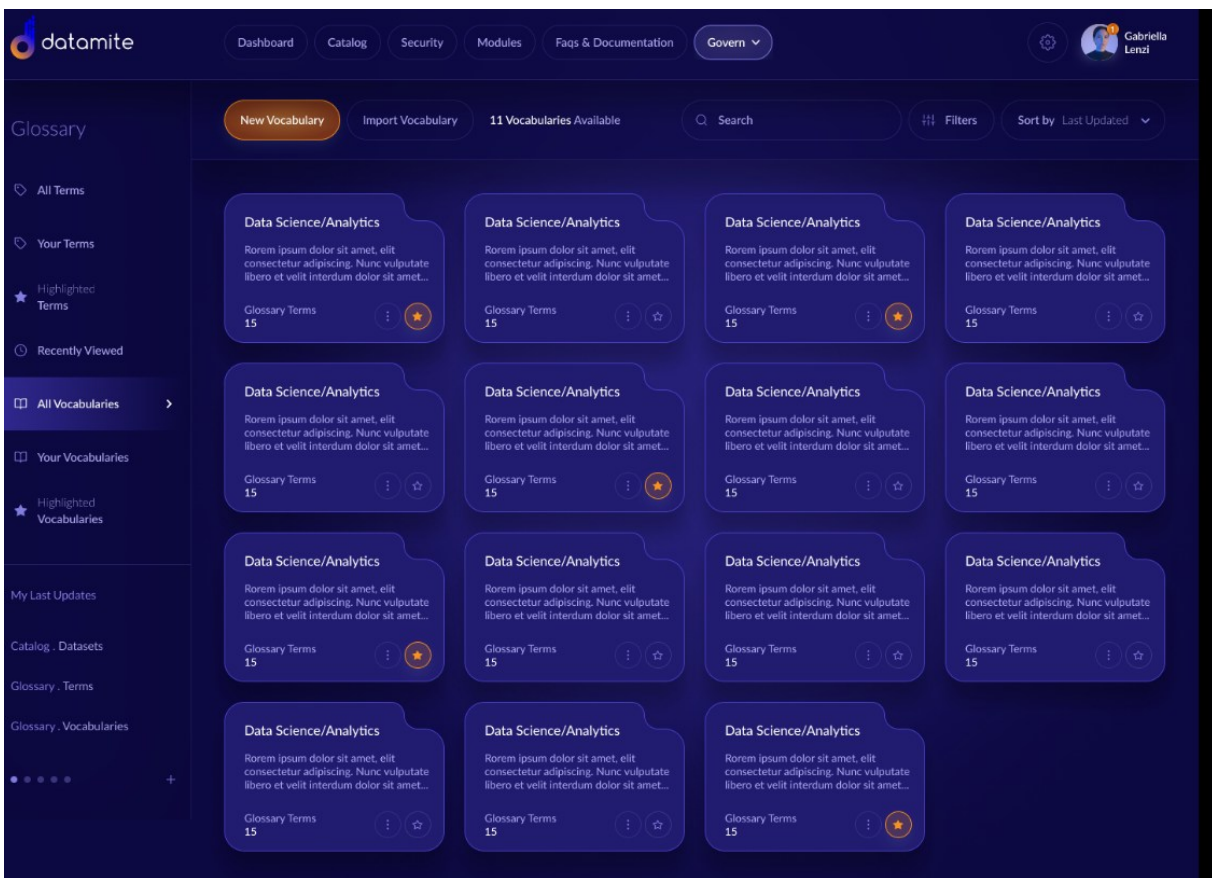


Figure 11: Vocabularies Page on the Glossary

The top bar presents several options, giving the user the option to see the “Your Vocabularies” page and the Highlighted Vocabularies. As shown in Figure 12, the content in the Domains page has a similar behaviour. It is possible to add a new domain, import it or use the search bar, filters and the sort by option. The cards display some details where the domain is associated with a number of artifacts, datasets, data products, terms and vocabularies. On the left side menu, it is possible to see the latest updates on the framework.

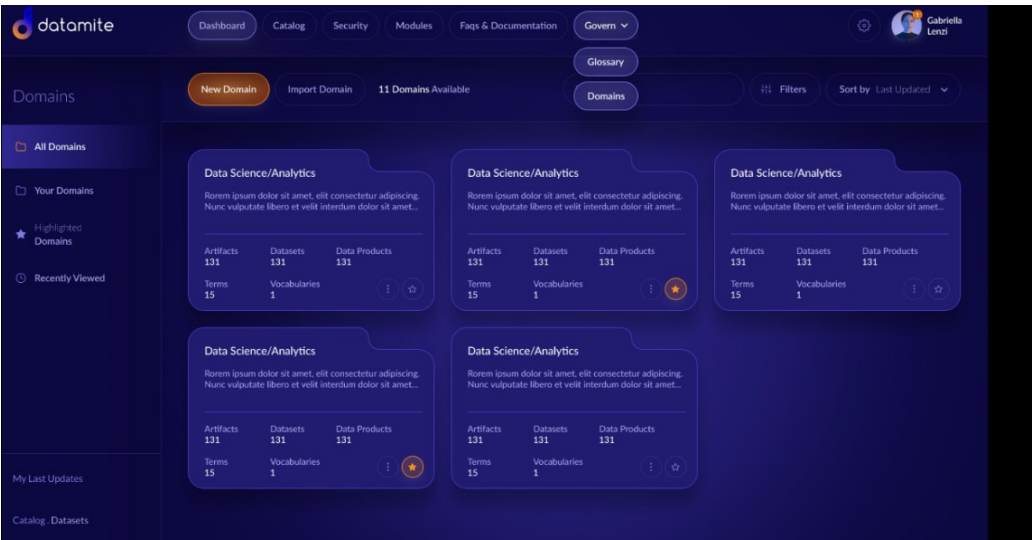


Figure 12: Domains Page on the Glossary

Figure 13 presents the Catalogue page, where the user can see the datasets available and can create new ones.

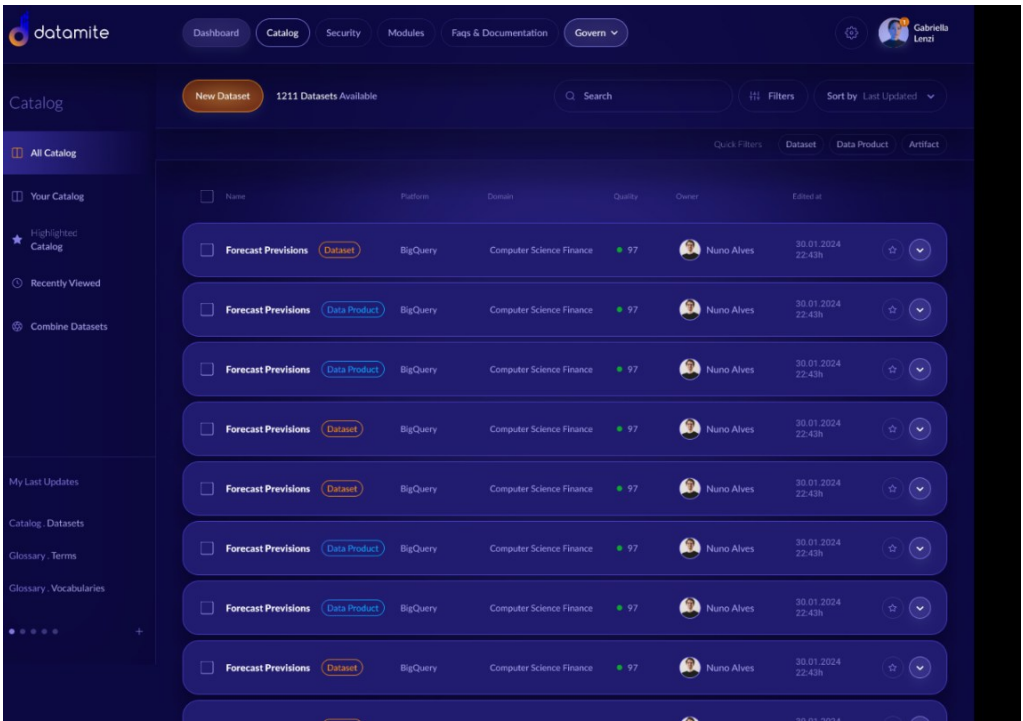


Figure 13: Catalogue Page



Three steps are required to create a new dataset: defining the dataset information, assigning permissions and finally confirming the creation. Note, however, as seen in the figures, that these screens are currently at the wireframe stage. Figure 14, Figure 15 and Figure 16 present the first step, the screens related to the information that can be requested from the user. In these screens, the user is required to provide information such as the name of the dataset, select the type of data to be added, e.g., bulk or streaming data, or provide basic and extended information from the user. This extended information spans from the type of file or a description to the linking with terms from a particular vocabulary.


Create New Dataset

1 Dataset Info

2 Permissions

3 Confirmation

Owner



Edgar Amorim
Analyst

Creation Date

24 Jan 2024

Dataset info

What should be the name of this dataset?


Customers

Method


Bulk

Streaming


Import one or more files



Import new file




Customers
.CSV



Payments
.CSV

Fill in with the basic info

Figure 14: Create a New Dataset – Type of data

 Funded by
the European Union

© DATAMITE 2023-2025

32/117




Create New Dataset

1 Dataset Info

2 Permissions

3 Confirmation

Owner

Edgar Amorim
Analyst

Creation Date

24 Jan 2024

Fill in with the basic info

Description

Norem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eu turpis molestie, dictum est a, mattis tellus. Sed dignissim, metus nec fringilla accumsan, risus sem sollicitudin lacus, ut interdum tellus elit sed risus. Maecenas eget condimentum velit, sit amet feugiat lectus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Praesent auctor purus luctus enim egestas, ac scelerisque ante pulvinar.

Format

CSV

JSON

XML

Excel Spreadsheet

SQL

SQLite

HDF5

Parquet

Avro

TFRRecord

Other

Platform

Q Search

AWS S3

Looker

BigQuery

AWS S3

Looker

BigQuery

AWS S3

Looker

BigQuery

AWS S3

Looker

BigQuery

Domains

Q Search

Mathematics

Computer Science/Artificial Intelligence

Computer Science/Finance

Medicine/Health Sciences

Civil Engineering

Computer Science/Linguistics

+312

Terms

Q Search

Convolutional Neural Network (CNN)

Epidemiology

Vocabularies

Q Search

Mathematics

Computer Science/Artificial Intelligence

Figure 15: Create a New Dataset – Basic Information


Create New Dataset

1 Dataset Info

2 Permissions

3 Confirmation

Owner

Edgar Amorim
Analyst

Creation Date

24 Jan 2024

Terms

Q Search

Convolutional Neural Network (CNN)

Epidemiology

Supply Chain Management

Regression Analysis

Blockchain Technology

Structural Engineering

+312

Vocabularies

Q Search

Mathematics

Computer Science/Artificial Intelligence

Computer Science/Finance

Medicine/Health Sciences

Civil Engineering

Computer Science/Linguistics

+312

Properties

We need info about this

Queries

We need info about this

Anonymisation

We need info about this

Validation


We need info about this

More info

We need info about this

Continue

Figure 16: Create a New Dataset – Extended Information

 Funded by the European Union

© DATAMITE 2023-2025

33/117



The goal of the second step, shown in Figure 17, is to manage the permissions for the dataset. Finally, the third and final step, which can be seen in Figure 18, is to validate the creation of the dataset. Tentatively, the frontend may show the quality score attributed to the dataset.

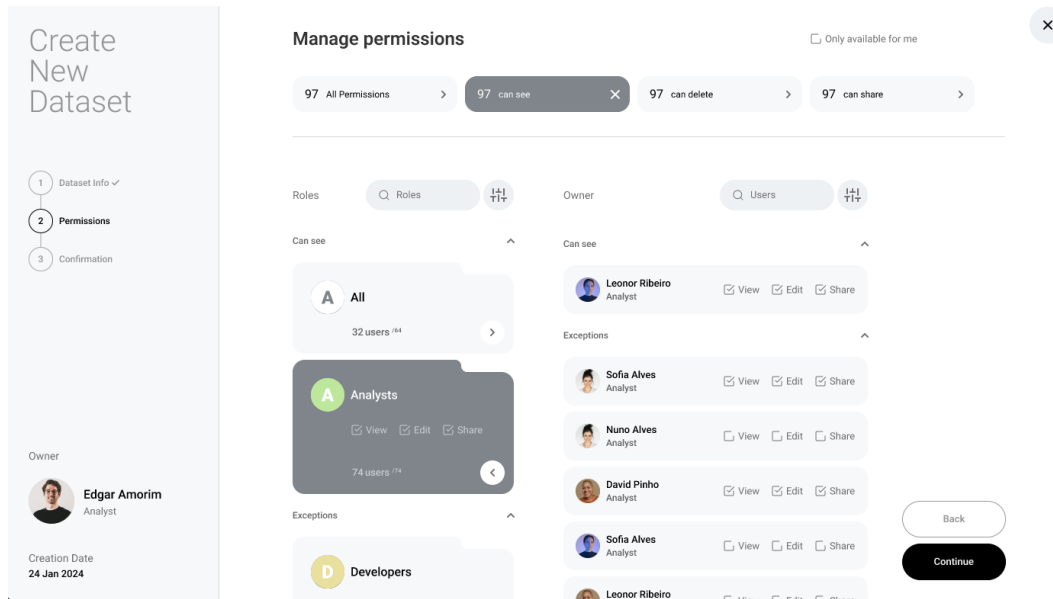


Figure 17: Create a New Dataset – Permissions

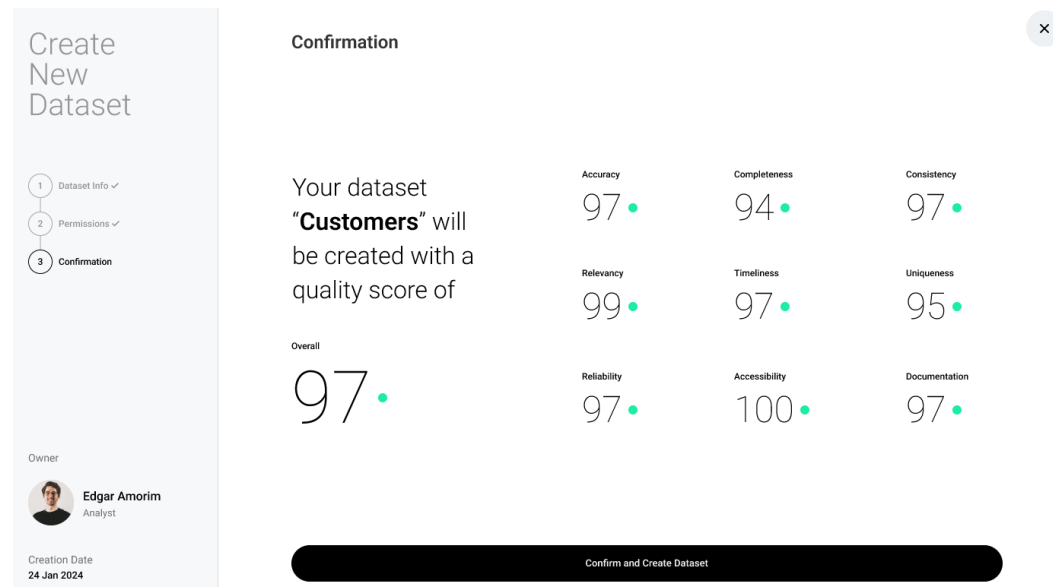


Figure 18: Create a New Dataset – Confirmation



In the listing on the catalogue page, the user can also see the details of the datasets and data products available and use quick filters for the different types of data. Figure 19 and Figure 20 present the detail page of a dataset, including basic information such as creation date, last edition, domain, format, etc.; and the details for its artifacts, including their size or linked terms, among others. Figure 21 presents information related to the different domains that can be used to tag the datasets. Similarly, in the data glossaries page, displayed in Figure 22, it is possible to add new vocabularies, new terms and their description or relations. Finally, as presented in Figure 23, the permissions tab allows for managing different kinds of permissions for the dataset, depending on the role of a profile.

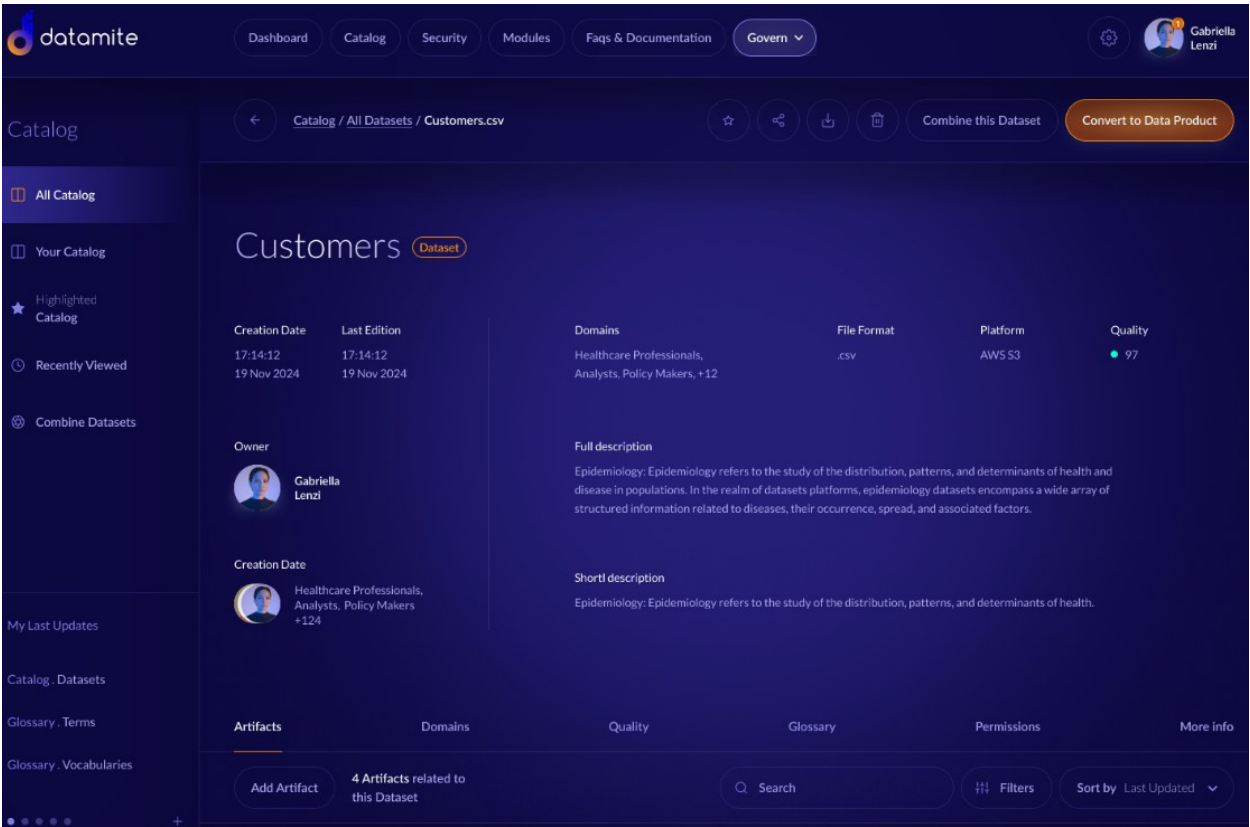


Figure 19: Detail Page of a Dataset

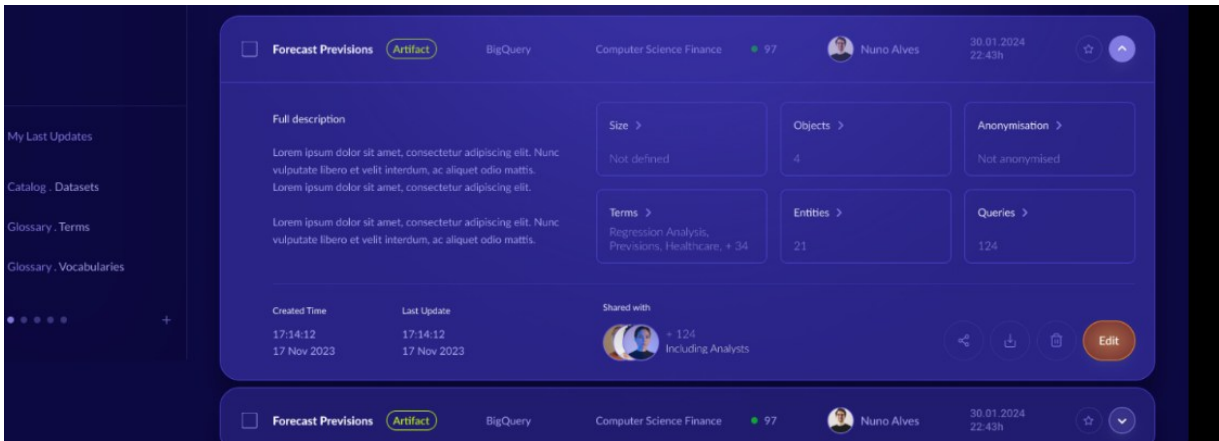


Figure 20: Detail Page of a Dataset - Artifacts

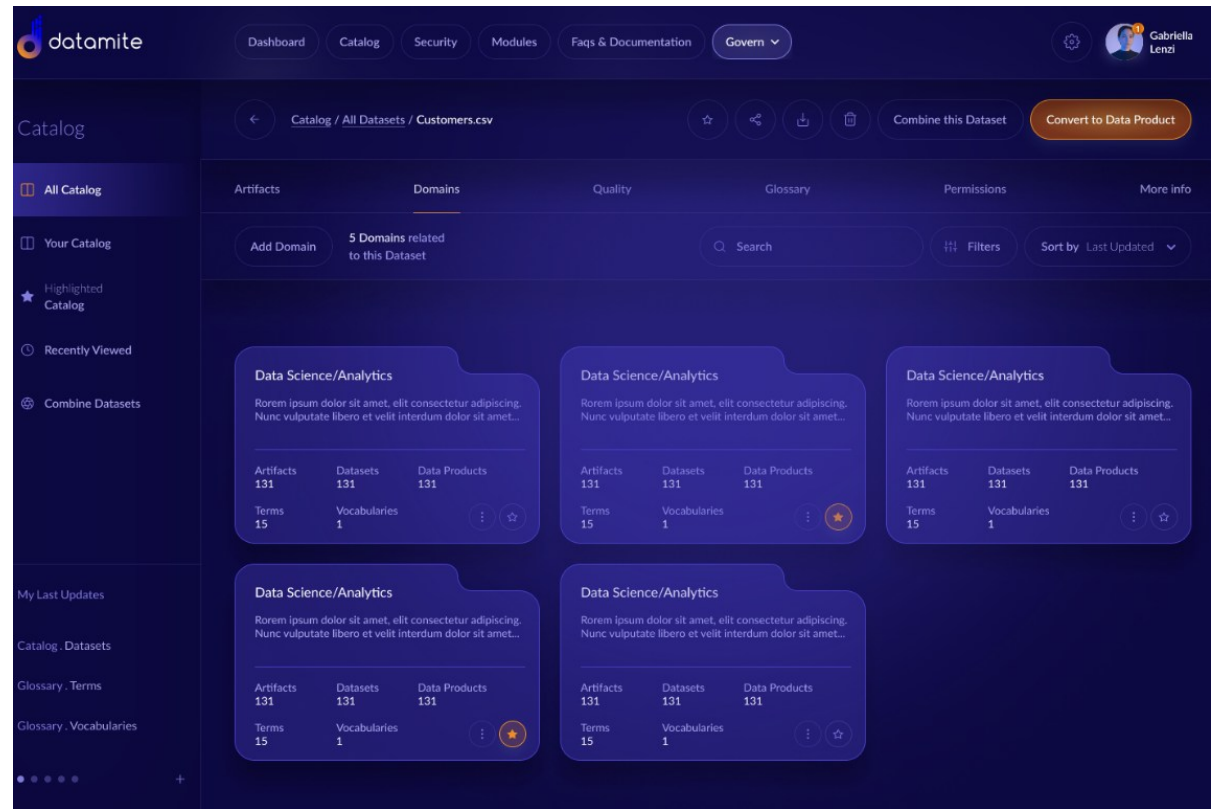


Figure 21: Detail Page of a Dataset - Domains

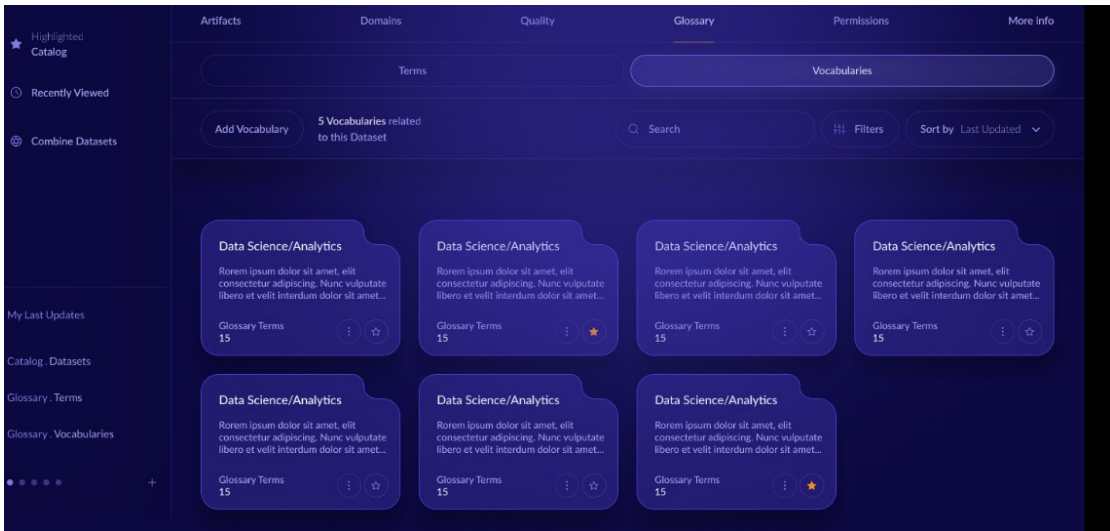


Figure 22: Detail Page of a Dataset - Glossary

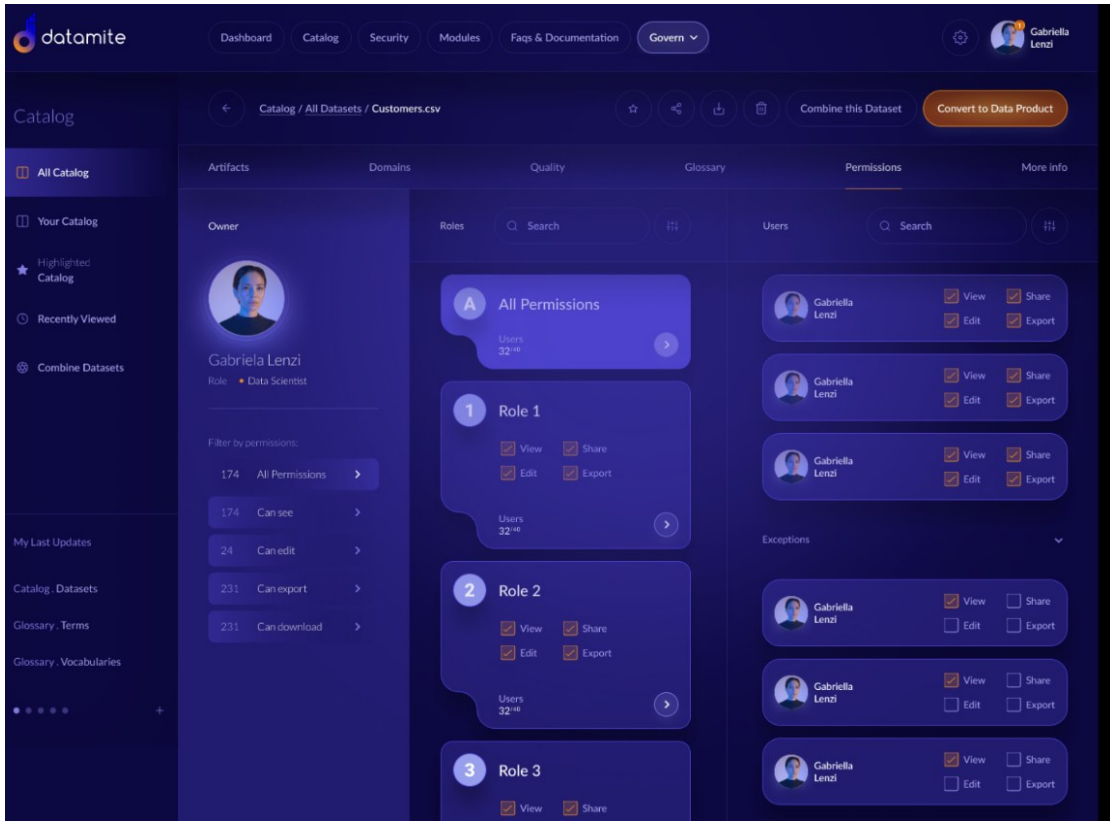


Figure 23: Detail Page of a Dataset - Permissions



2.3.3 Final Status

Some of the most relevant advances regarding these components are related to:

- Integration with Access Control. Although this cannot be visualised per se, the components have been integrated with the access control component, hence, only showing certain options, if available, for the user who is logged in. This has also involved the creation of the login page, as can be seen in Figure 24.
- Dashboard. The dashboard has also evolved to a nearly final version -some adjustments may result from the interaction with the pilots-, giving much more protagonism to the datasets, data products or publications. A view can be seen in Figure 25.
- Data Catalogue view. This screen has evolved to include more comprehensive information about the datasets or to trigger the creation of data products. Moreover, this development has been taken as a basis to have a separate view for the data products. These interfaces can be seen in Figure 26 and Figure 27.
- Dataset creation. The interfaces required for the creation of datasets are in place, having different steps where the most relevant would be the initial description, which actually creates the corresponding entity in the Metadata Repository, that can be seen in Figure 28; or the selection of data sources, that shows a different interface depending on the data origin, as exemplified in Figure 29 and Figure 30 for bulk files and database queries.
- Dataset information and tabs, with the different metadata that can be edited by the user or that is generated during the ingestion process. An example can be seen in Figure 31.
- Data Dictionary details for data quality. It contains the information regarding the inherent or user-defined quality KPIs computed by the Quality Evaluator for each data artifact.
- Filters on listings to facilitate the search for entities. This implied enabling filters for the platform, domain, quality, owner or date when a dataset was edited.
- Present relations between terms (synonyms and antonyms) to facilitate semantic interoperability.

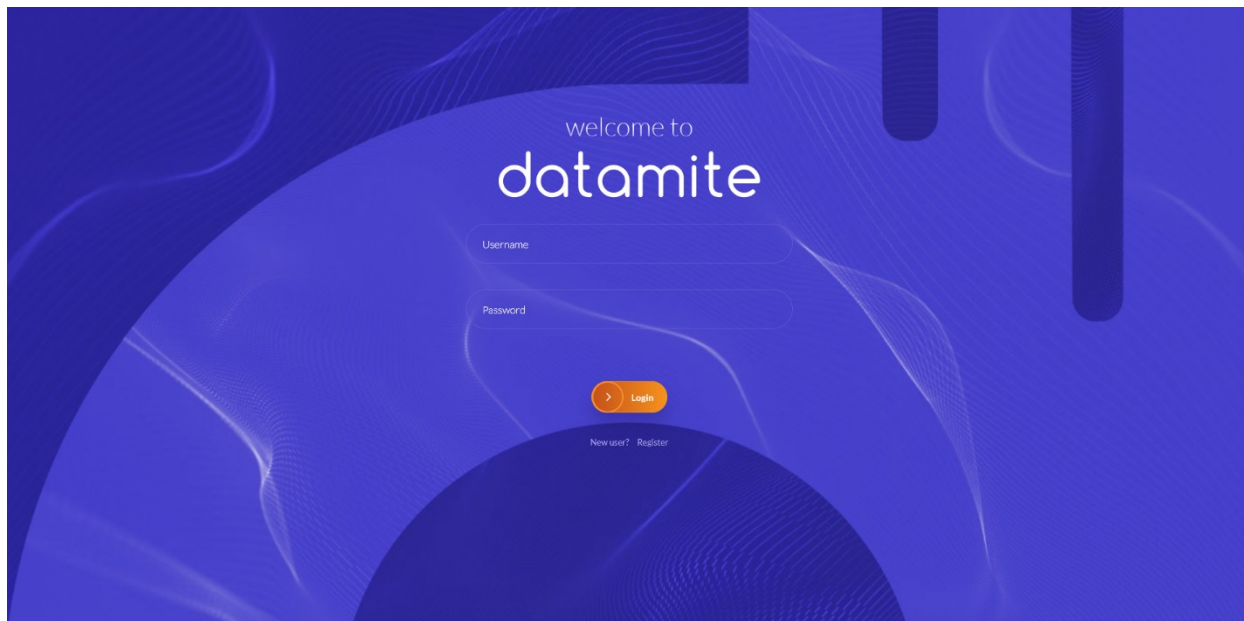


Figure 24: Login page

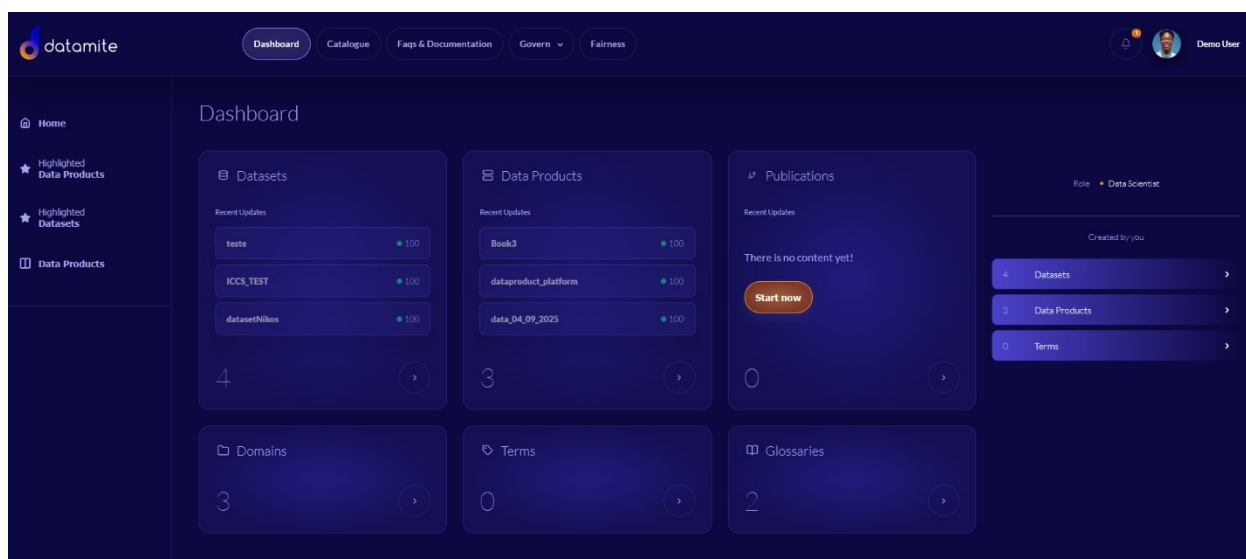


Figure 25: Dashboard



<input type="checkbox"/>	Name	Domain	Quality	Edited at	
<input type="checkbox"/>	teste	dataset	97	Sep 10 2025 10:37:05	
<input type="checkbox"/>	ICCS_TEST	dataset	97	Sep 12 2025 10:54:07	
<input type="checkbox"/>	datasetNikos	dataset	97	Sep 10 2025 17:57:07	
<input type="checkbox"/>	datasetPanos	dataset	97	Sep 10 2025 17:07:07	

Figure 26: Dataset catalogue view.

<input type="checkbox"/>	Name	Domain	Keywords	Anonymization
<input type="checkbox"/>	Book3	dataprodukt	laboris ex nostrud inure, eiusmod ullamco fugiat proi	Id consequat, nostrud sed
<input type="checkbox"/>	dataprodukt_platform	dataprodukt	laboris ex nostrud inure, eiusmod ullamco fugiat proi	Id consequat, nostrud sed
<input type="checkbox"/>	data_04_09_2025	dataprodukt	laboris ex nostrud inure, eiusmod ullamco fugiat proi	Id consequat, nostrud sed

Figure 27: Data products catalogue view



Create
New
Dataset

1

Dataset Info

2

Method

3

Permissions

4

Confirmation

1. Dataset name

2. Fill in with the Basic Info

Full description

Short description

Domains

ArtificialIntelligenceMathematicsAnalytics

Glossary/Terms

☐ Forecast Previsions

☐ Blockchain

Create

Figure 28: Creation of datasets.



Create
New
Dataset

1 Dataset Info

2 Method

3 Permissions

4 Confirmation

1. Method

Bulk

Streaming

Database Connection

Queries

File name

File type

Date format

%Y-%m-%d

Import the file

Import new file

Back

Create

Figure 29: Creation of datasets. Data sources - Bulk.



Create New Dataset

1 Dataset Info
2 Method
3 Permissions
4 Confirmation

1. Method

Bulk Streaming Database Connection **Queries**

Select a Database Connection

Select from previous usage

Name

Is Recurring

True False

Start Date dd/mm/aaaa End Date dd/mm/aaaa Interval Hours

Back Create

Figure 30: Creation of datasets. Data sources - DBQuery.

TestCSV **Artifact**

Creation date Sep 12 2025, 06:54:49	Last Edition Sep 12 2025, 06:54:49	Domains -	File Format -	Quality -
Full description -				
Short description -				

Content Components Schema Data Sample Lineage Properties Queries Validation Anonymisation Quality Rules Glossary Permissions More Info

1 Artifact(s) related to this artifact

Q Search Filters Sort by Last updated

Columns	Num rows	Duplicate entries	Duplicate entries (%)	Completeness	Completeness (%)	
KPIs						
User Defined Rules						

Figure 31: View of a dataset



2.4 Other components

The Data Governance Module has two more components regarding Data Lineage and Versioning, as shown in Figure 32. However, the development of these two components had not started by month 18. Even when depicted as components in the architecture, these functionalities are provided embedded in other components or the frontend. The detail for each of them is described in the following subsections.

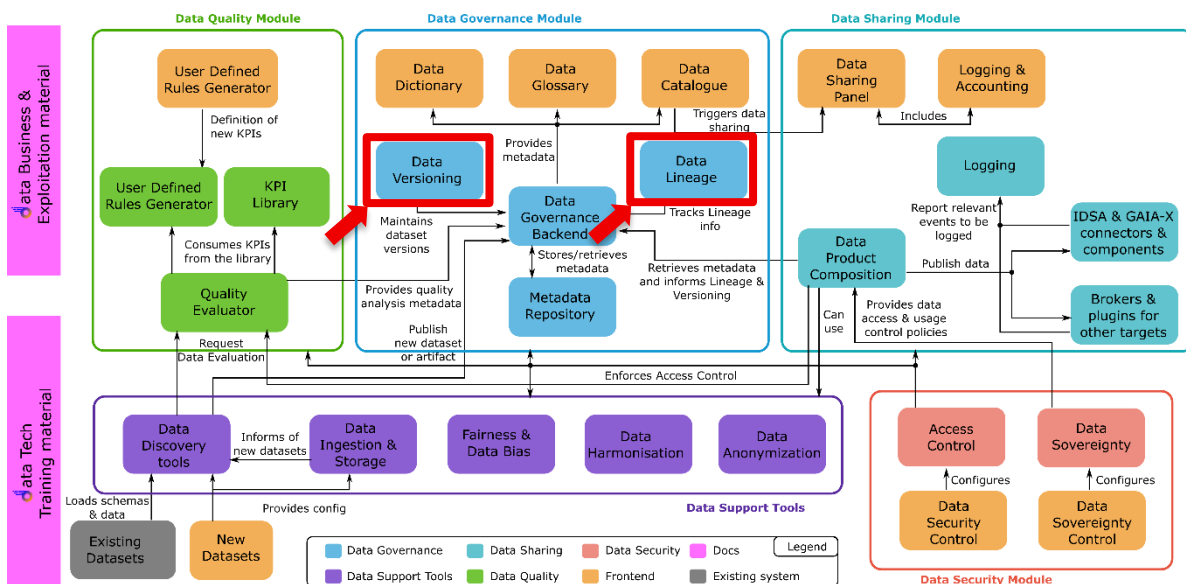


Figure 32: Data Lineage and Versioning in the architecture

2.4.1 Data Lineage

The goal of the Data Lineage component is to display the evolution of data sources since they are brought into the framework. Hence, it is especially relevant for the creation of datasets and artifacts when they relate to an existing data source, such as a database or a data stream; or for data products, to provide a traceability indicating what datasets were used to create a data product.

Therefore, to capture this lineage, a series of APIs -those related to the creation of different kinds of artifacts and similar entities- have been enriched to capture the information needed for the



```
"lineage": {  
  "process": {  
    "title": "string",  
    "description": "string",  
    "action": "string",  
    "executionDate": "2025-09-16T10:10:08.683Z"  
  },  
  "sources": [  
    "3fa85f64-5717-4562-b3fc-2c963f66afa6"  
  ]  
},
```

Figure 33: API extension to creation APIs (Data product, structuredFiles, etc.) to include lineage.

lineage. The structure seen in Figure 33 shows the information being captured, including the title of the process or operation being performed, a description, the corresponding action or actions performed on the data, the execution date, and the data sources that were taken as a base to create the resulting entity.

Additionally, given its complexity, there is an alternative extension for the creation of DBQuery artifacts, that can be seen in Figure 34. This structure is similar but includes additional fields to capture the potential recursiveness of these kinds of entities. It would include, as an extra field, the *query* used to extract the data from the database, the *queryId*, the *recursive* field to indicate if this is a recursive query and, in case it is, the *start* and *end times* during which the query will be executed every *intervalHours*.

These extensions have been implemented and integrated during the different creation processes of the relevant entities.

Figure 34: API extension to the DBQuery API to include lineage

The goal of Data Versioning is to allow users to recover different versions from a data product stored in the framework. The dataset artifacts are not considered as files that user will be editing. The expected behaviour is that if an artifact needs to be modified, it will be reuploaded into the framework.

To do so, the intermediate files are stored separately, adapting their paths and metadata to reflect the existence of different versions of the same artifact. The recovery of these previous versions is possible through the lineage data interface, where they can be visualised and downloaded. Finally, GET APIs that return data product artifacts have been modified to list only the last versions of artifacts or all of them, as both functionalities are required in different moments. Examples of these listings can be seen in Figure 35 and Figure 36.



```
"distributions": [  
  {  
    "id": "cf801d50-875a-4e73-be12-86159fc0b1f2",  
    "name": "artefacto dp 2",  
    "type": "StructuredDistribution",  
    "qualifiedName": "73c50c5b-a062-4a8c-a512-ff31b794b5f2",  
    "relationId": "1dacdfa0-615e-4c3c-a41c-5c81f8989420",  
    "score": null,  
    "createdAt": "2025-09-16T14:12:17.201Z",  
    "lastVersion": true  
  }  
],
```

Figure 35: Get API returning all versions from an artifact.

```
"distributions": [  
  {  
    "id": "c728048f-35a2-45c0-adb6-386d656178f7",  
    "name": "artefacto dp 1",  
    "type": "StructuredDistribution",  
    "qualifiedName": "51d9dc31-d404-4ef2-9c3f-e2d36a6ef5d2",  
    "relationId": "d1a82dc5-a378-44c3-bd39-f482a1512000",  
    "score": null,  
    "createdAt": "2025-09-16T14:12:00.114Z",  
    "lastVersion": false  
  },  
  {  
    "id": "cf801d50-875a-4e73-be12-86159fc0b1f2",  
    "name": "artefacto dp 2",  
    "type": "StructuredDistribution",  
    "qualifiedName": "73c50c5b-a062-4a8c-a512-ff31b794b5f2",  
    "relationId": "1dacdfa0-615e-4c3c-a41c-5c81f8989420",  
    "score": null,  
    "createdAt": "2025-09-16T14:12:17.201Z",  
    "lastVersion": true  
  }  
],
```

Figure 36: Get API returning all versions from an artifact (multiple versions exist).

3 Data Quality Module

This section presents the status of the Data Quality Module components which encompass the Quality Evaluator, KPI Library, and User Defined Rules Generator (frontend and backend).

3.1 Quality Evaluator

The Quality Evaluator (QE) is a component that belongs to the DATAMITE's Data Quality module, as shown in Figure 37, designed to evaluate a dataset from the quality point of view. It receives data from the discovery tools and profiles it, evaluating specific indicators based on user configurations. The quality evaluation is the result of computing different indicators, inherent or user defined, available in the KPI Library or the User Defined Rules Generator backend. This results in the generation of informative profile templates that can be accessed and utilised via the platform's frontend catalogue, providing users with actionable insights and quality-related metrics.

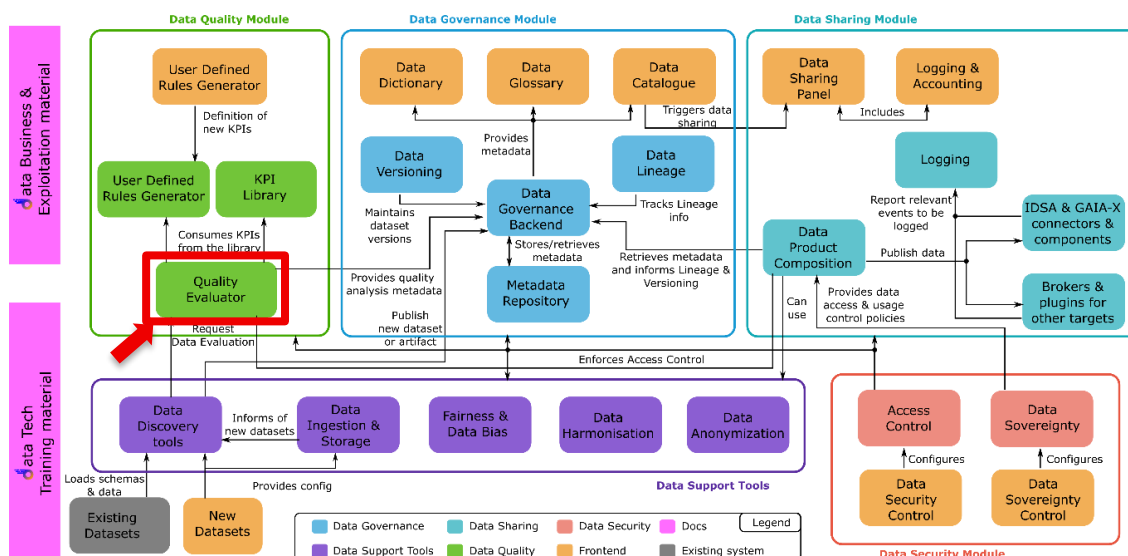


Figure 37: The Quality Evaluator in the architecture

3.1.1 Design Overview

The QE processes data coming from the Data Discovery tools. The Evaluator is a software tool implemented using Python, which awaits an API request to begin its operation. This API indicates where the data to be processed is stored, so it can be fetched. Once the data has been loaded,



a Spark job is initiated, completing the process of profiling and user-generated rules application to the dataset. The use of Apache Spark as the infrastructure framework was chosen as it enables the Evaluator to process both small and large volumes of data, although a sampling approach should be adopted in the latter case.

The data received is forwarded through the QE's Data Retrieval layer to the Spark job. Then it undergoes pre-processing, filtering and preparing data for detailed analysis. For this step, the Python code is informed about the types of the received dataset's columns or, otherwise, computes them. Thus, it is feasible for the QE to match the KPIs of the KPI Library (and later of the User Defined Rules Generator – UDRG) to the proper columns, based on their types. Although some data types (e.g., integers, text) can be identified by the Python code, there are more complex types (e.g., timestamps and timestamp formats) that can lead to errors in the KPI evaluation phase. For this reason, information about the data type can be received along with the dataset. This approach increases the probability that the generated results are of high quality.

The pre-processed data is subsequently fed into the KPI evaluation phase, leveraging the KPI Library and User Defined Rules Generator, i.e., the other two components of the Data Quality module, as seen in Figure 37. At first, the component parses a library of predefined KPIs to identify which metrics are applicable to the dataset. Based on the data types of each column in the retrieved dataset, the Spark job performs a matching of the KPIs to the dataset's columns. For example, KPIs that refer to numerical columns are only applied to the columns specified as numerical. This process allows for the automatic extraction of meaningful quality analytics, providing insights and trends that are essential for informed decision-making. Once all the KPIs have been applied to the dataset, the generated analytics are collected in a dataset profile template, which is forwarded to the Data Governance Backend of the DATAMITE framework.

In addition to the KPIs in the KPI Library, users can define custom KPIs through the User Defined Rules Generator. When an end-user defines their own rules through the DATAMITE platform's UI, these rules are stored in the User Defined Rules Generator. At the same time, the Quality Evaluator is triggered through a new Spark job. The Evaluator invokes the User Defined Rules Generator and consumes the new quality analytics extracted from the dataset. It then returns

those insights to the Data Governance Backend, like the automatic KPI evaluation phase. This capability allows for tailored analytics that meet specific DATAMITE user needs and objectives.

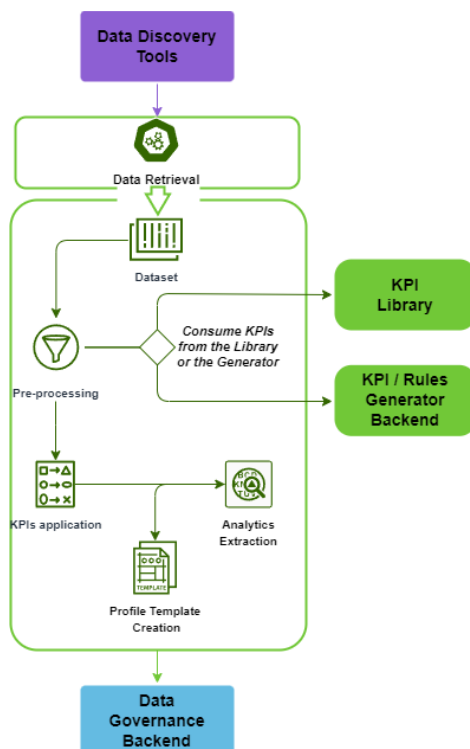


Figure 38: The Quality Evaluator's operational flow

In conclusion, the Quality Evaluator component exemplifies a simple approach to data retrieval and analytics. Through the Evaluator, both automatic and user-defined KPI applications to datasets are achieved. The software exploits two other components (or libraries) of the Data Quality module, the KPI Library and the User Defined Rules Generator, to provide data profiling and analytics extraction functionalities to the end-users within the DATAMITE platform.

3.1.2 Status at M18

By the end of M18, the Quality Evaluator's main infrastructure and operation were implemented. A standalone Apache Spark Cluster was set up, along with two Spark jobs which were fully functioning. One job integrates with the KPI Library, while the other integrates with the UDRG.



3.1.3 Final Status

By the end of M33, the development of the QE component progressed, taking into consideration the changes made to the other two components of the Data Quality module (i.e., the KPI Library and the User Defined Rules Generator backend) as well as two other components of the DATAMITE platform, the Data Discovery and Data Governance Tools. In the final version of the Quality Evaluator, the latest KPI Library was used, which provides metrics not only for structured data but also for unstructured data such as images, audio, and video. Moreover, the Quality Evaluator was updated to reflect changes made to the User Defined Rules Generator backend. Finally, the interface changes in the Data Discovery and Data Governance Tools, which directly affect the Quality Evaluator, were considered.

More specifically, in the final version of the Quality Evaluator, significant improvements were made both in terms of functionality and reliability. The mechanism for profiling definitions was reworked so that they can now be retrieved dynamically from the User Defined Rules Generator (UDRG) through API calls, ensuring greater configurability and tighter integration into the DATAMITE platform. At the same time, several critical bug fixes and optimisations were introduced, leading to more stable execution, improved handling of concurrent operations, and the merging of previously overlapping jobs. Another important enhancement was the integration with the latest version of the KPI Library, which extended the Quality Evaluator's profiling capabilities beyond structured datasets. With this integration, the component can now process unstructured data such as images, audio, and video. Artifacts are first identified and categorised by MIME type, after which the corresponding KPIs are applied to extract quality-related characteristics. Finally, the results are consolidated into aggregated dataset-level profiles, ensuring consistent evaluation of both structured and unstructured data within the same framework.

In parallel, important work was carried out to adapt the Quality Evaluator for deployment in containerised environments. Dedicated health check endpoints were added to support liveness and readiness monitoring, facilitating smooth integration with Docker-based setups and paving the way for Kubernetes-native deployment. Furthermore, the storage layer for cross-container communication was migrated from the local filesystem to a MinIO-based object storage solution,



aligning the component with modern cloud-native practices and enabling scalability in large-scale data processing scenarios. The Quality Evaluator was designed to assess large-scale datasets with high efficiency. Built on top of Apache Spark, it can seamlessly process massive volumes of data when the necessary infrastructure is available, ensuring scalability and robustness. At the same time, it offers flexibility through data sampling, enabling faster evaluations when there is no urgent need for full dataset precision. The final version of the Quality Evaluator has been uploaded to the [Eclipse Repo⁷](https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-quality/quality_evaluator).

3.2 User-Defined Rules Generator

The User Defined Rules Generator (UDRG) is a component of the Data Quality module. It allows the users of the DATAMITE framework to define business or context-aware rules that can provide specific indicators and values considered relevant within the company or use case.

Considering the necessity to interact with the user, the creation of rules must be a process guided by the DATAMITE frontend, offering an editor for this purpose. The user would select the available indicators, taken from the KPI Library, as well as the columns that the given dataset has in its schema. Once the rules are created, the evaluation of the quality, performed through the integration with the QE, will use them to produce the quality metadata related to the dataset or the data element that will be evaluated.

The definition of the rules follows a formal representation based on the DDQV (DATAMITE DQV⁸); vocabulary extended from the DQV to cover the full expressivity of the rules as the UDRG presents them. The custom metrics are the central element from which the expressions can be formed, combining several simple expressions using logical operators. DDQV is an extension of DQV, which is being developed within the context of this task.

⁷ https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-quality/quality_evaluator

⁸ <https://www.w3.org/TR/vocab-dqv>



3.2.1 Design Overview

The Data Quality module is composed of the UDRG, the KPI Library, and the QE, which is part of the general DATAMITE architecture and is depicted in Figure 39. The UDRG comprises backend and frontend components, in green and orange, respectively. The interactions of the Data Quality module with the rest of the architecture are principally with Data Governance, to access and store the quality metadata, and with the Data Discovery, Ingestion and Storage tools to profile the data. The workflow and the elements involved in the process of creating a rule and their evaluation are reflected in the sequence diagrams shown in Figure 40 and Figure 41.

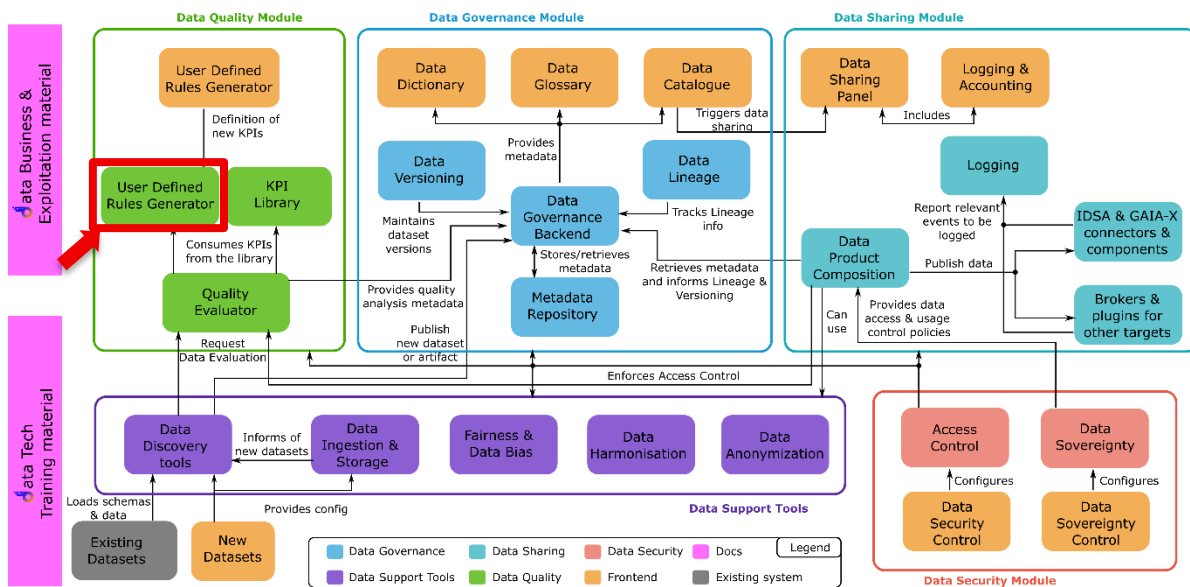


Figure 39: Data Quality Module in the DATAMITE Architecture

The elements involved in the rule creation process are the frontend or GUI, the Metadata Repository (ATLAS), the KPI Library and the rules library (a MongoDB database internal to the UDRG). The GUI is in charge of presenting to the user the elements to be selected (from a given dataset) as operands and operators so the final expression can be as complex as required. Also, the possibility of redefining the thresholds for a specific rule is provided from the GUI. The storage of the rules is performed as a call to a method provided by the UDRG API. The user can reuse rules as they are available for selection and modification. Some adjustments should be performed to follow the schema for the current dataset.

The process interpreting the rules is called by the QE once the KPIs and the rules involved are informed by invoking a method provided by the UDRG API. Hence, the QE calls the interpreter passing the values of the calculated KPIs that are needed for the rules, the *dataset_id* and the data to be evaluated as parameters.

Rules creation

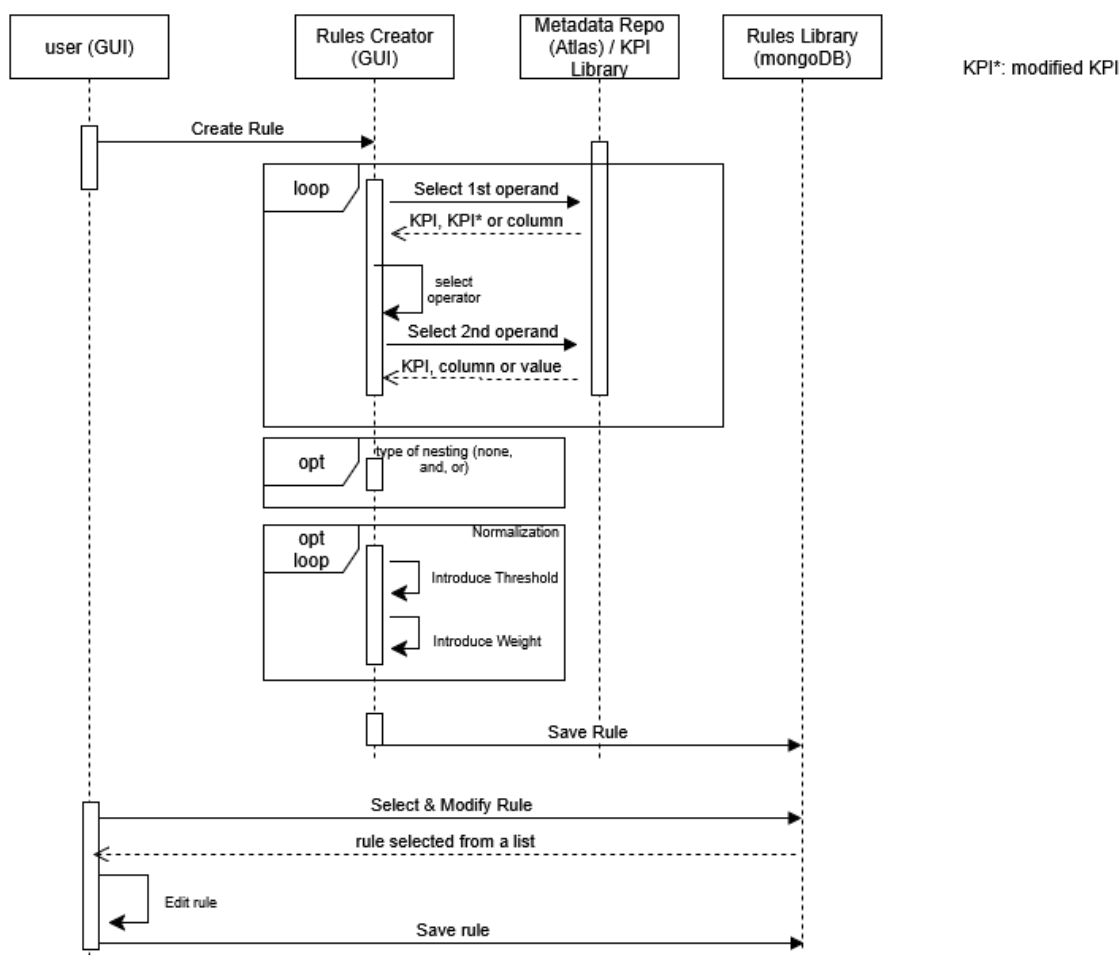


Figure 40: Rules creation sequence diagram

The result (normalised if thresholds and weights are introduced) of this interpreter is formatted according to the input format expected by the QE and delivered. The QE comprises the metadata obtained when calling the KPI Library and UDRG to fulfil the metadata model followed in the metadata repository. In addition to the processes already presented, a model for representing the

quality metrics has been designed. The DDQV, presented in the Figure 42 has been developed as an extension of the DQV vocabulary.

Rules interpretation (Evaluator)

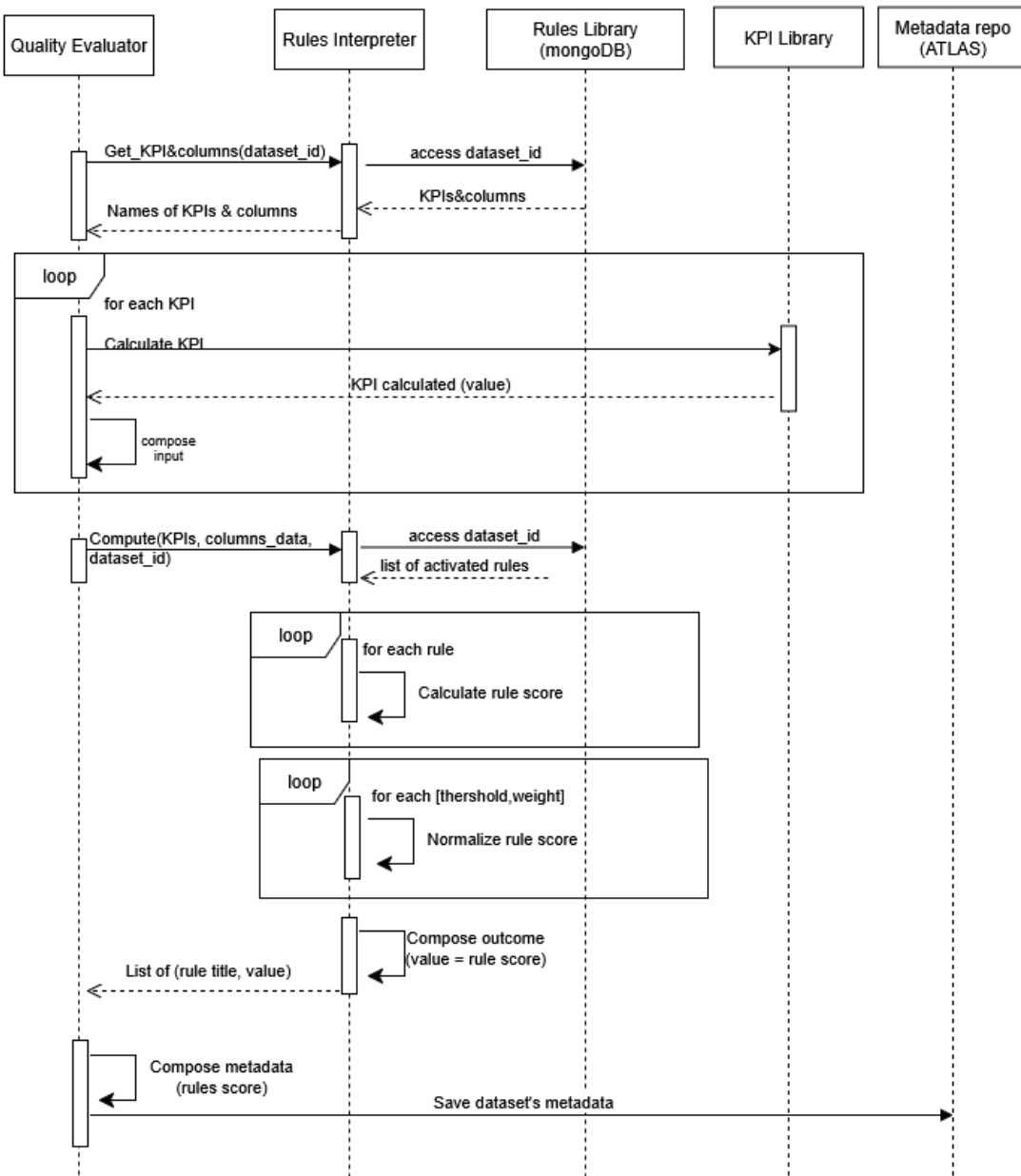


Figure 41. Rules interpretation sequence diagram

ddqv: Datamite DQV

The relationship `ddqv:refersTo` from `ddqv:LeftOperand` class can have as range `ddqv:Metric` or `skos:Concept` type of classes, so it can use as leftOperand the "ColumnValue", besides `Metric`.

Instances of `ddqv:Operator` class would be: `gt`, `lt`, `eq`, `gteq`, `lteq`, `in`, `regexMatch`.

Instances of `ddqv:LogicalOperator` class would be: `or`, and

`ddqv_datFormat` is an optional property, and is used to specify the date format in case the field it refers to is a date.

`ddqv_opModifier` is an optional property and is used to modify the value of the field it refers to, by using the arithmetic expression specified in this property.

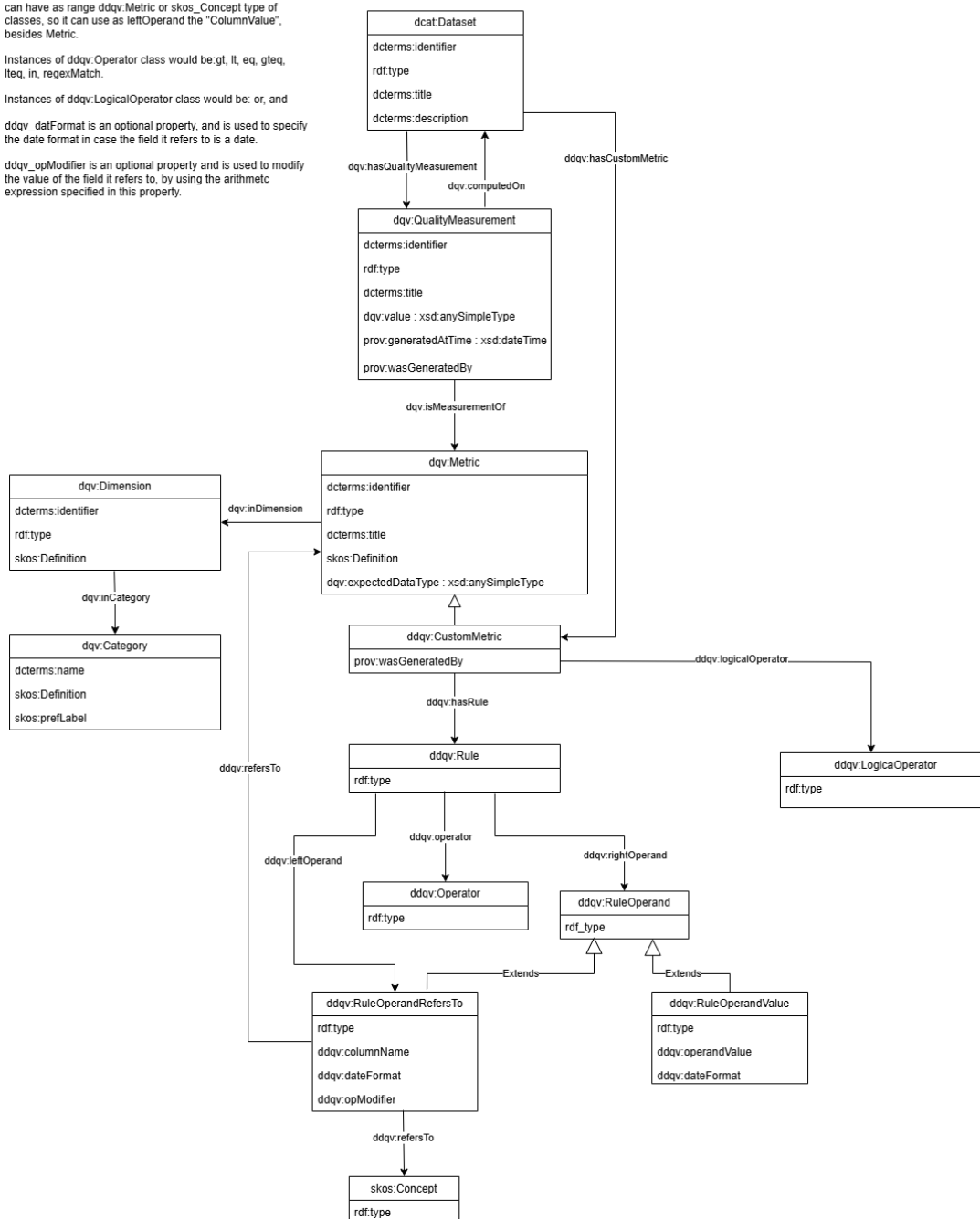


Figure 42: DATAMITE Data Quality Vocabulary schema



3.2.2 Status at M18

The developed functionalities via API were:

- Get all custom metrics: for getting the rules stored in MongoDB.
- Get custom metrics associated with a dataset: for getting the rules associated with a given dataset.
- Add/update a custom metric for a dataset: to store the custom metric created in the MongoDB.
- Get the required KPIs for the custom metrics: for getting the KPIs involved in the custom metrics related to a specific dataset. This method is used by the QE to calculate the KPIs before invoking the UDRG.
- Get the required columns for the custom metrics: the QE needs to know the data to pass to the UDRG for evaluating the custom metrics.
- Evaluate the custom metrics associated with a dataset: the central method of the UDRG in charge of calculating the values of the custom metrics associated with a specific dataset, using the previously obtained KPIs.
- Get a description of a custom metric: extracting an understandable description of a custom metric and interpreting the expression behind it.
- Delete a custom metric of a dataset.
- Delete all custom metrics of a dataset.
- Get the status (active/not active) of custom metrics associated with a dataset.
- Update (change) the status of custom metrics associated with a dataset.

Moreover, the definition of the GUI needed for creating the rules has been done as a preliminary version and is under integration with the Data Quality Module frontend, whose status is described in Section 3.4. Likewise, the computation of the different rules is available via API, and the integration with the QE is defined and described in Section 3.1 - Quality Evaluator.

The database for storing the custom metrics is already created and available. In particular, it is a MongoDB database, which consists of a collection of documents representing the custom metrics associated with a dataset.

The definition of the Data Quality model is in its second version, covering specifically the formal expression of the “custom metrics”. By this definition, the rules are the simple expression of a custom metric, allowing the possibility to increase the complexity of those simple rules using logical connectors (AND/OR), in which case the element in the model corresponds to the custom metrics.

3.2.3 Final Status

The final version is available in the [Eclipse Repo](https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-quality/user-defined-rules-generator)⁹. Figure 43 presents the internal architecture of the UDRG component.

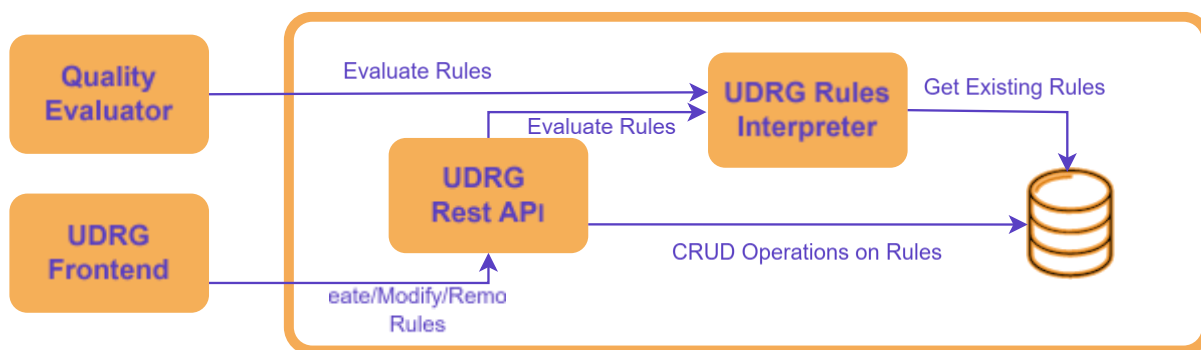


Figure 43: UDRG architecture

The front end provides the end user with an editor to create the rules. These rules are created by selecting the available indicators, taken from the KPI Library, as well as the columns that the given dataset has in its schema. This editor invokes the UDRG Rest API, which creates, modifies or removes the rules in the rules database.

Once the rules are created, their evaluation will be carried out by the QE, which will invoke the UDRG Rules Interpreter through the UDRG Python library provided. The final version of this module includes the functionalities described in previous sections and new ones, some of which

⁹ <https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-quality/user-defined-rules-generator>



belonged to the “Next Steps” paragraph in the corresponding section of the D3.2 deliverable. The developed functionalities are listed below, along with their final status:

- *Suggest a quality dimension for a custom metric: to help the user with the classification of the rule/custom metric.* This functionality was not implemented. This attribute of the rule is part of the description of the rule, not necessary to evaluate it; therefore, the knowledge about the quality the user has is enough to select one of the proposed dimensions.
- *Calculate quality level for the specified dimension: envisioned for the general score of the quality in the context of the DATAMITE frontend. The definition would be related to the needs of the users when sharing or using the data, which means designing the final score to present on the corresponding page and the contribution to this score, involving the participants in every sub-module of the Data Quality Module.* The design of this functionality has evolved. Considering the level of quality as a general score, the quality needs of the users are covered by the definition of specific rules that evaluate the quality by applying those specific requirements to the data. However, the general quality score calculation has been improved by developing a method for normalising the evaluation result of each rule, giving it the weight that the user considers it should have when computing that final general score. The final functionality developed provides the evaluation result normalised, using threshold-weight values configured by the end user.
- *Convert rules into RDF format: for a particular use of the custom metrics for sharing the metadata of a dataset or data product when the Gaia-X data spaces are involved.* This functionality is implemented and part of the final version, and available to be used in case of further versions of the DATAMITE framework, but is not currently presented in the FE
- *Get dataset columns: It would give the possibility to access the metadata repository to retrieve the schema of the dataset when the columns are not known.* This service was removed from the list of new developments since the columns are provided by the Data Governance Backend API.
- *Analysis and definition of a profiling configuration in the form of a file or a structure are under study. The KPIs and rules the user wants or needs to evaluate in a given moment of the DATAMITE workflow should be presented for selection and subsequent consultation by the QE.* The Front end presents to the user the possibility of disabling or enabling the



rules and, therefore, deciding which rules will be evaluated when the profiling process is launched. This is an attribute stored as part of the rule information.

The work related to the UDRG is tightly connected to the Data Quality module, so collaboration among the technical developments is highly necessary. One of the collaborations thoroughly conducted has been the frontend definition of the UDRG, involving not only this module but the functionalities provided by the DQM.

Finally, the discussions with the use cases were very useful, and from them several excellent ideas emerged that were included in the development of the module. To take those requirements into consideration, the DDQV model was updated with new elements and the UDRG code was adapted according to their suggestions.

In particular, the new implementations supported by the UDRG module and reflected as options in the Front End were:

- It is possible to express the date format of an operand when the field to be evaluated requires this information.
- Sometimes the operand involved in the rule needs to be modified, not taking it as its literal value, so the numeric operands and the date ones can be changed by applying arithmetic operations, like a percentage or adding/subtracting a value.
- New operators are available for selection: “In” operator for rules that need to know if a field of the data is in a list of values, and “RegexMatch” operator to be used when the comparison the rule needs to make is if a field matches a regular expression pattern.

The UDRG module is currently under integration and the final pilots’ evaluation process. Any issues and modifications needed from that process will be described in the corresponding deliverable if necessary.

3.3 KPI Library

The main goal of this component is to implement inherent quality metrics that allow us to obtain relevant information (metadata) about different types of data, regardless of the context in which they will be used, so that they are as generic as possible. In short, the goal of the library is to



implement methods that allow users to profile data, allowing the QE service to consume them efficiently.

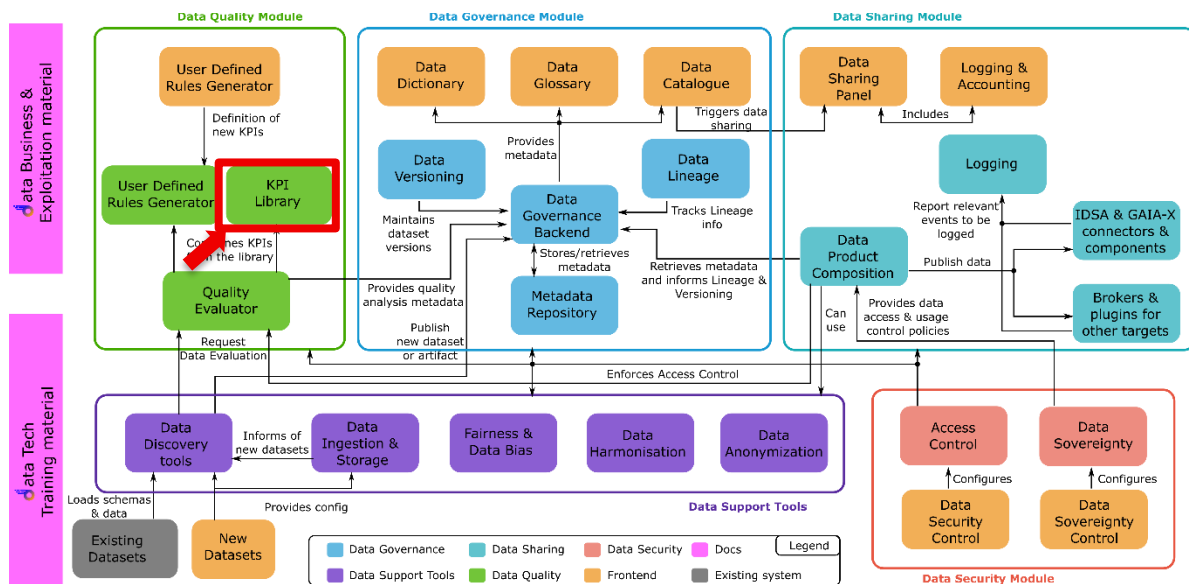


Figure 44: The KPI library in DATAMITE architecture.

Additionally, this component returns the metadata in a way that makes the system interoperable with other platforms. For this purpose, the standard DQV (meta)data model will be used within the library. Figure 44 presents the KPI library in its context.

3.3.1 Design Overview

As mentioned in the introduction, this library provides different metrics that allow different profiling pipelines to be carried out on the data. Data profiling is the process of examining, analysing, and summarising datasets to understand their structure, content, and quality. It involves collecting statistics and informative summaries about the data, which help in identifying data quality issues (missing values, outliers, correlations, etc), understanding data distributions, and ensuring the data is suitable for analysis or other downstream tasks.

When working with data, it is important to recognise the different types of data that might be encountered:



- **Structured Data:** This type of data has a standardised format, typically tabular with rows and columns that clearly define data attributes, like databases and spreadsheets (CSV, XLS, etc). Additionally, each column has a specific data type (e.g., integers, floats, dates or strings).
- **Semi-structured Data:** Semi-structured data does not adhere to a strict schema but contains organisational properties, such as tags or markers to separate data elements. Some examples will include JSON or XML files.
- **Unstructured Data:** Unstructured data lacks a predefined format or organisation, making it more challenging to analyse directly, such as text documents, images, or videos.

Different data types require different profiling techniques because each type has unique characteristics and challenges. For example, structured data is straightforward to profile using various statistical and visualisation techniques, which help understand its central tendency, variability and relationships, whilst profiling semi-structured data often involves converting it into a structured format to apply standard profiling metrics effectively. Thus, it is necessary to extract and understand the schema of the data, to check the consistency of the key-value pairs and to identify missing or incomplete fields within the semi-structured data. Lastly, profiling unstructured data involves specialised techniques to extract useful information, such as Natural Language Processing, like tokenisation or named entity recognition, or Image Analysis, like pattern recognition or feature extraction.

3.3.2 Status at M18

At M18, the KPI library contained profiling metrics for structured data, and these metrics were grouped according to the type of data they process: general metrics, analyse any type of data; and specific metrics for numerical, categorical, date, and text data. The metrics for specific data only profile individual columns, while the general metrics can process individual columns or the entire dataset.

The results of the metrics are returned following the DQV standard. Specifically, this module focuses mainly on two DQV components, Metric and Quality Measurement (both are shown in Figure 42). The Metric component provides the information related to a particular quality indicator, and when this indicator is used to profile any dataset, it returns a value that is stored in a



QualityMeasurement object. Each component uses different properties to describe itself. On the one hand, the Metric component utilises:

- *skos:definition* property to define what the metric does.
- *dqv:expectedDataType* property to refer the expected data type of the score obtained after applying this metric.

On the other hand, the Quality Measurement component employs:

- *dqv:isMeasurementOf* property to refer the metric the score comes from.
- *dqv:computedOn* property to refer to the resource on which the quality measurement is performed.
- *dqv:value* property to encode the observed value of the metric, i.e., the result or score after applying a metric on a dataset.

To implement DQV in the KPI library, a method called `to_dqv` has been developed in each class, which is responsible for returning the result as a QualityMeasurement object. To be more precise, the method will return the code shown in Figure 45.

```
{
  "dqv_isMeasurementOf": "class_name.metric_name",
  "dqv_computedOn": "data_name",
  "dqv_value": "result_of_processing_the_metric_in_the_data",
  "ddqv_hasParameters": [{
    "parameter_name": "parameter_name",
    "value": "value_of_the_parameter"
  }]
}
```

Figure 45: Object structure obtained from the `to_dqv` method.

The “*ddqv_hasParameters*” key is a new property, created for the purposes of this project, to store the values of the parameters that have been used to implement that metric in the data. For example, one of the implemented metrics is the histogram, which allows the user to specify the number of columns with which they want to build the histogram. The “*ddqv_hasParameters*”



property does not appear in the model predefined by DQV, although discussions were held on how to represent the parameters of the metrics.

Returning to the `to_dqv` method, this form of implementation has been decided because it allows the quality process to be carried out knowing only the type of data to be profiled, the name of the metric to be implemented and the value of the parameters. Using the previous example, if the QE needs to run the histogram metric to analyse numeric data, it will use the code in Figure 31. Figure 46.

```
from kpi_library import NumericMethods
NumericMethods.to_dqv(
    method_name="histogram",
    parameters=[{"parameter_name": "num_bins", "value": "5"}]
)
```

Figure 46: Example of implementation of the histogram metric.

3.3.3 Final Status

The KPI library has been extended to include methods for non-structured data, namely, images, audio or video. The approach is extendable, only needing to add more KPI evaluation methods to the library that will be automatically detected for DATAMITE. The internal structure of the library can be seen in Figure 47.

Similarly, on deployment time or as per user demand, the KPI library is scanned by a script that extracts the currently available evaluation methods in the library and pushes this information into a database. The database contains all the descriptive information about the different methods, classified by the type of data they can be used with. The information in the database can be consumed by any other component that may require listing the available KPIs, such as the User Defined Rules Generator frontend.



For more information, the README file, which can be found in the [Eclipse Repo¹⁰](https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-quality/kpi-library), provides all the information related to each metric implemented in the library.

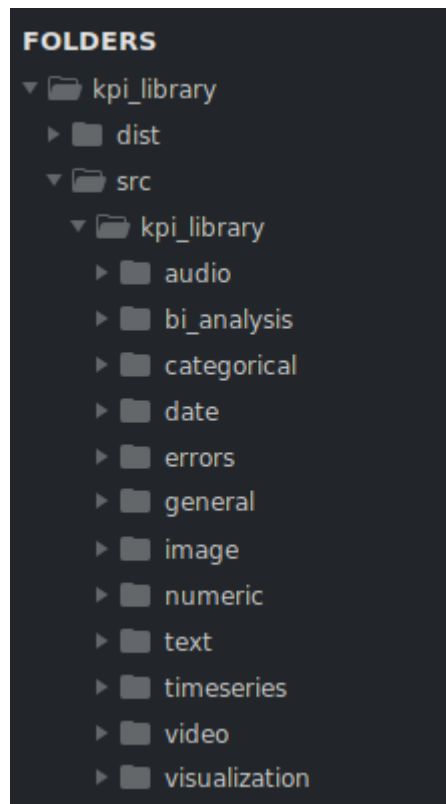


Figure 47: KPI Library internal structure

3.4 Data Quality Module frontend

Regarding the data quality module, the focus is on the implementation of a dataset profiler (considering aspects related to data accessibility, security, privacy, constraints on data formats and storage capacity) and mechanisms to define specific rules integrated in the governance

¹⁰ <https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-quality/kpi-library>



components. This section describes the efforts associated with the development of the frontend, while Figure 48 presents the components where the Quality frontend is involved.

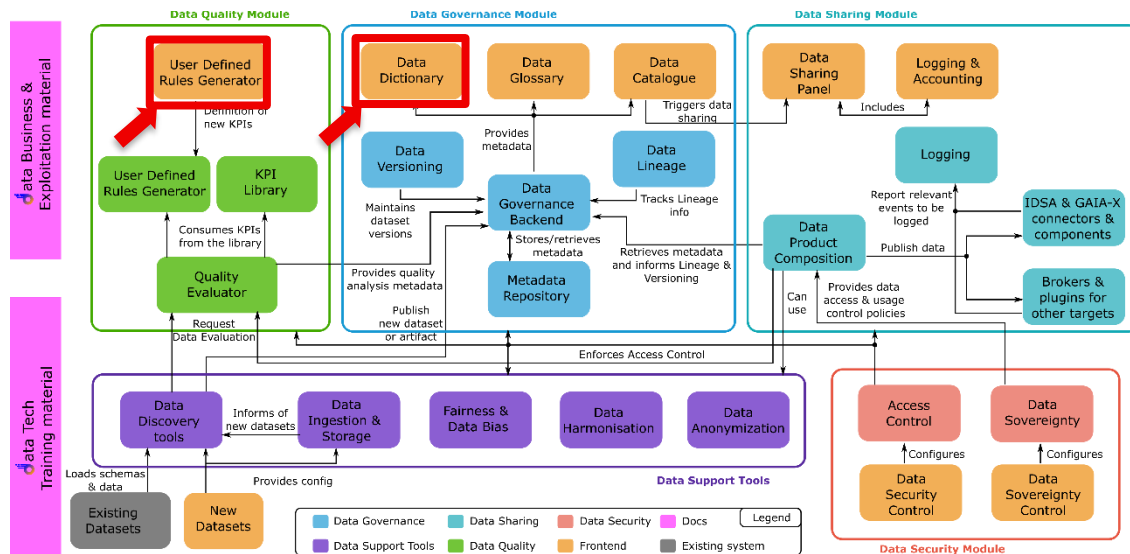


Figure 48: The Data Quality frontend in the DATAMITE architecture.

3.4.1 Design Overview

Besides the Data Dictionary, already described in Section 2.3, there are some quality aspects offered by the frontend. First, the ingestion of a dataset or the preparation of a data product involves the need of profiling, evaluation of rules and computation of an overall quality score to that specific dataset, as seen in Figure 49. The quality of the dataset also includes several individual dimensions, such as Consistency, Accuracy, Completeness, Timeliness, Uniqueness, Validity, among other possibilities.

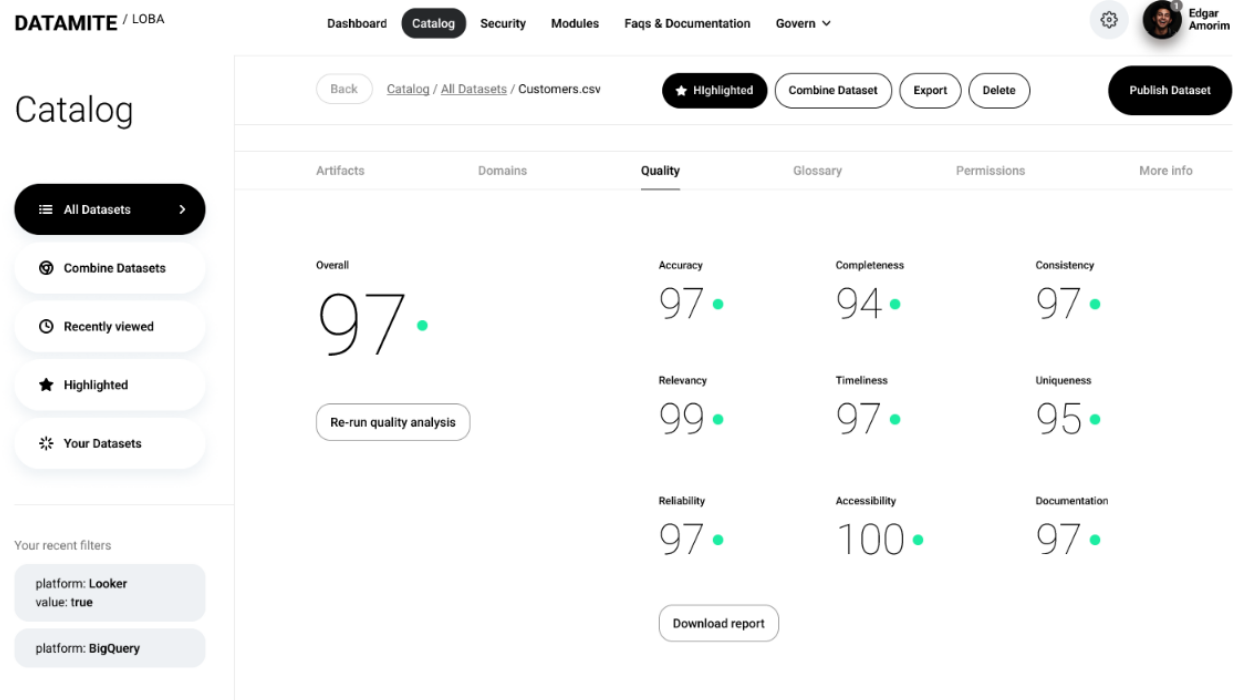


Figure 49: DATAMITE Data Quality Score, Catalogue view

Secondly, when the user is on the catalogue page and opens a dataset, one of the tabs that is available is the “Quality” tab. This component, besides showing the metrics previously shown on the creation of the dataset, has the possibility to re-run a quality analysis and/or download a quality report, as seen in Figure 49. Similarly, it will allow for the definition of user-defined quality rules. The “Re-run quality analysis” is intended to work as an on-the-moment update of the metrics generated, which may change depending on the data that may be altered in the datasets, such as the addition or removal of artifacts, and removal of domains, among other examples. The “Download report” option is a button with a call-to-action for the user to download a generated quality report based on the metrics presented, and it could include more analytical information, such as graphics and tables.

3.4.2 Status at M18

This quality score shows several dimensions with numerical metrics that evaluate each dataset and give it an overall score. At M18, however, most of the functionalities related to the data quality



module had only been designed on the wireframes. Its development was to be undertaken in the following months, both in the UI design and in the frontend code.

3.4.3 Final Status

The bulk of the work during this period has been devoted to preparing the interface for the UDRG, including the preparation of rules, as well as presenting indicators like the general quality score. These developments have gone through the different phases of wireframes, Figma design and implementation. Some of the main efforts have been:

- Developments regarding the creation of new rules. A view of the main interface for the step-by-step creation of a rule can be seen in Figure 50 and Figure 51.
- Developments regarding the visualisation of available or existing rules already created by the user. Examples of this interface, with AND or OR rules, and with thresholds to define the overall score, can be seen in Figure 52 and Figure 53.
- Developments regarding the listing of rules defined for an artifact are shown in Figure 54.
- Evaluation of the overall score of an artifact, as can be seen in Figure 55.

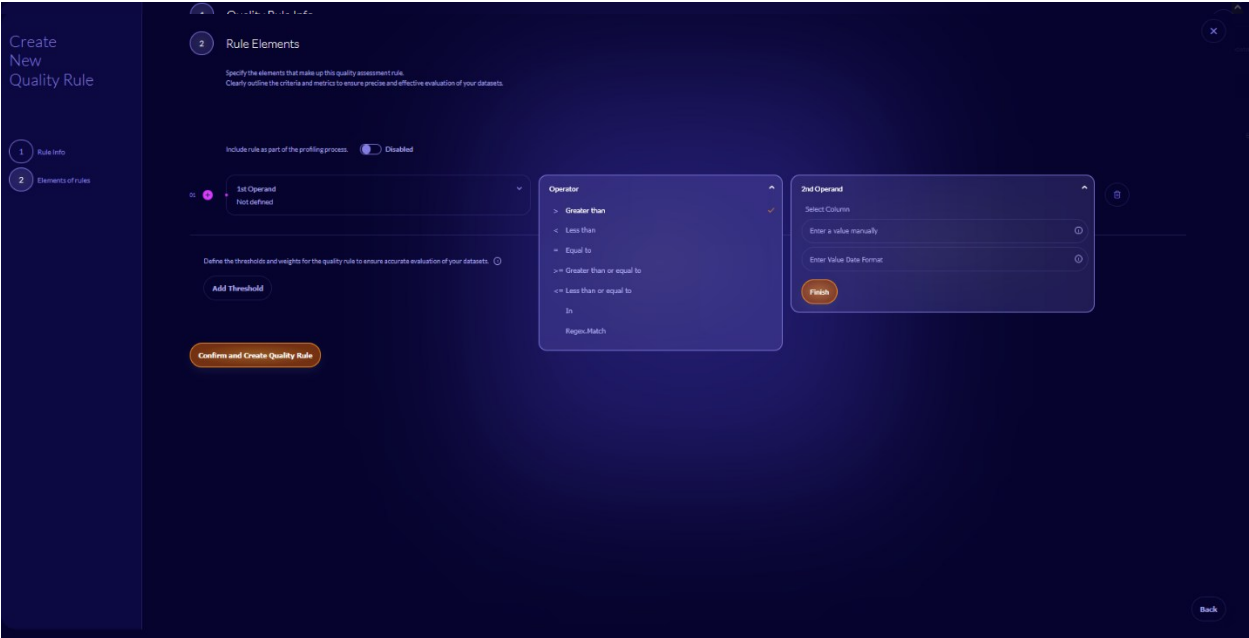


Figure 51: View of the Create rule interface (2).



Creation date: 17/14/12, 19 Nov 2024 | Last Edition: 17/14/12, 19 Nov 2024 | Dimensions: Consistency | Applied to: 241 Datasets | Owner: Demo User | Full description: Accepted values for total installed power

Rule elements | Related Catalog | Domains | Permissions | More info

Include rule as part of the profiling process: ☒ Enabled

01. 1st Operand: total installed power (W) / Column Value | Operator: >= Greater than or equal to | 2nd Operand: *10000***not double

02. 1st Operand: total installed power (W) / Column Value | Operator: <= Less than or equal to | 2nd Operand: *200000.0***not double

Define the thresholds and weights for the quality rule to ensure accurate evaluation of your datasets. ☐

Add Threshold

Save Changes

Figure 52: Example of rule defined with the UDRG.

Owner: Demo User | Full description: Year column accepted values

Rule elements | Related Catalog | Domains | Permissions | More info

Include rule as part of the profiling process: ☒ Enabled

01. 1st Operand: year / Column Value | Operator: >= Greater than or equal to | 2nd Operand: *2024***not integer

02. 1st Operand: Not defined | Operator: Not defined | 2nd Operand: Not defined

Define the thresholds and weights for the quality rule to ensure accurate evaluation of your datasets. ☐

Threshold: 0 | Weight: 0 | Threshold: 25 | Weight: 10 | Threshold: 50 | Weight: 30 | Threshold: 80 | Weight: 50 | Threshold: 100 | Weight: 100

Save Changes

Figure 53: Example of rule defined with the UDRG specifying thresholds.

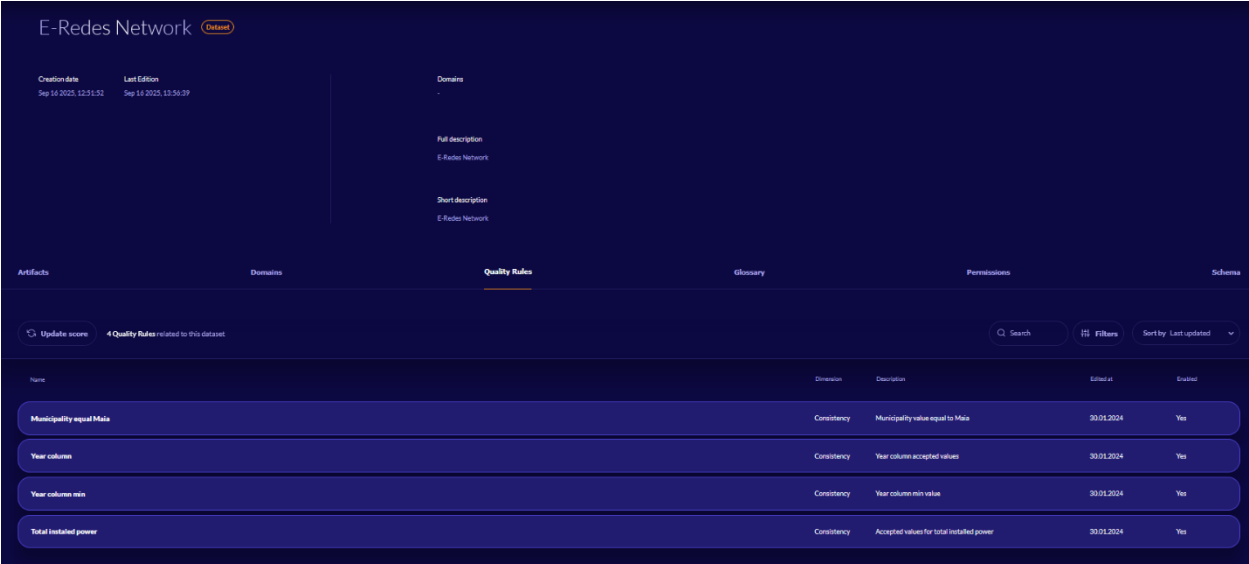


Figure 54: List of rules available in a dataset.

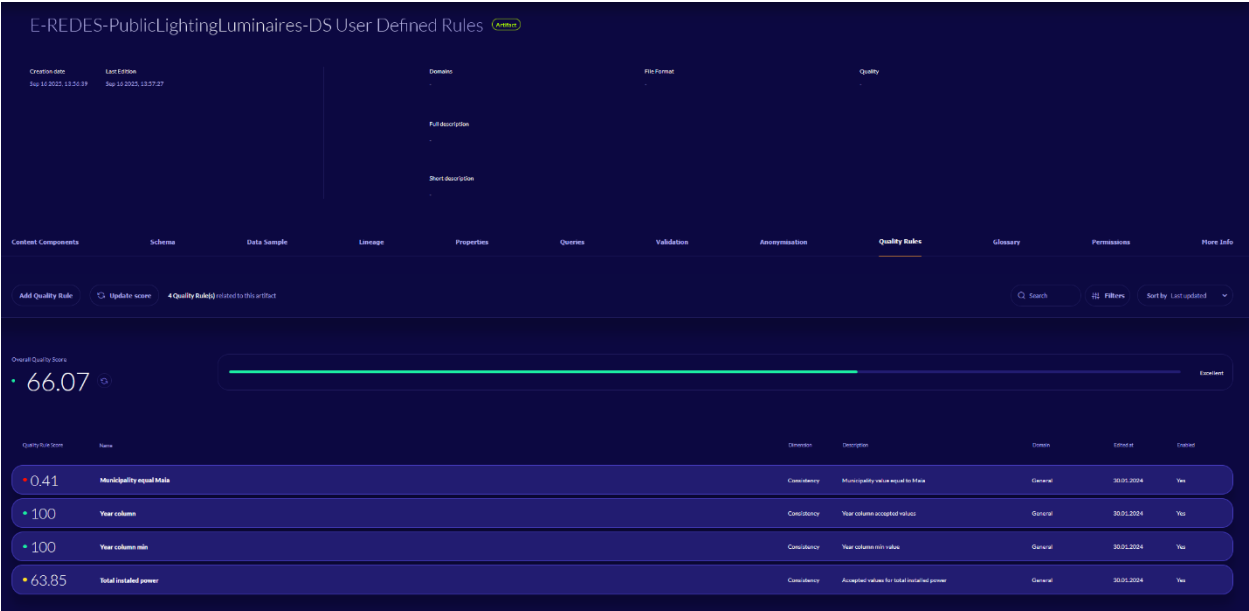


Figure 55: List of rules executed in an artifact and overall score.



4 Data Security Module

This section presents the status of the Access Control component in the Data Security Module regarding its implementation for the final code release. Its design, previous and final status are presented.

4.1 Access Control

Figure 56 depicts the Access Control component within the context of DATAMITE's architecture. In more detail, the Access Control interacts with all the modules: Data Quality, Data Governance, Data Sharing, and Data Support tools, to ensure fine-grained access control within the framework. The Access Control component uses Keycloak¹¹, functioning as a robust authorisation system that simplifies internal identity management and token-based access control within the DATAMITE framework. It is also utilised for user and account management.

4.1.1 Design Overview

This section provides an overview of the Access Control component's design. Its main purpose is to manage internal identities, ensuring a secure environment where user credentials are protected. The token-based access control mechanism ensures that access rights are granted via verifiable tokens, securely encapsulating the user's identity and permissions.

Additionally, the component supports various user management functions, including login processes, user registration, and deletion. The aim is to implement an effective authorisation strategy using Role-Based Access Control (RBAC). Permissions are assigned based on roles, which are then linked to users, ensuring a clear and manageable permissions structure.

¹¹ <https://www.keycloak.org/>

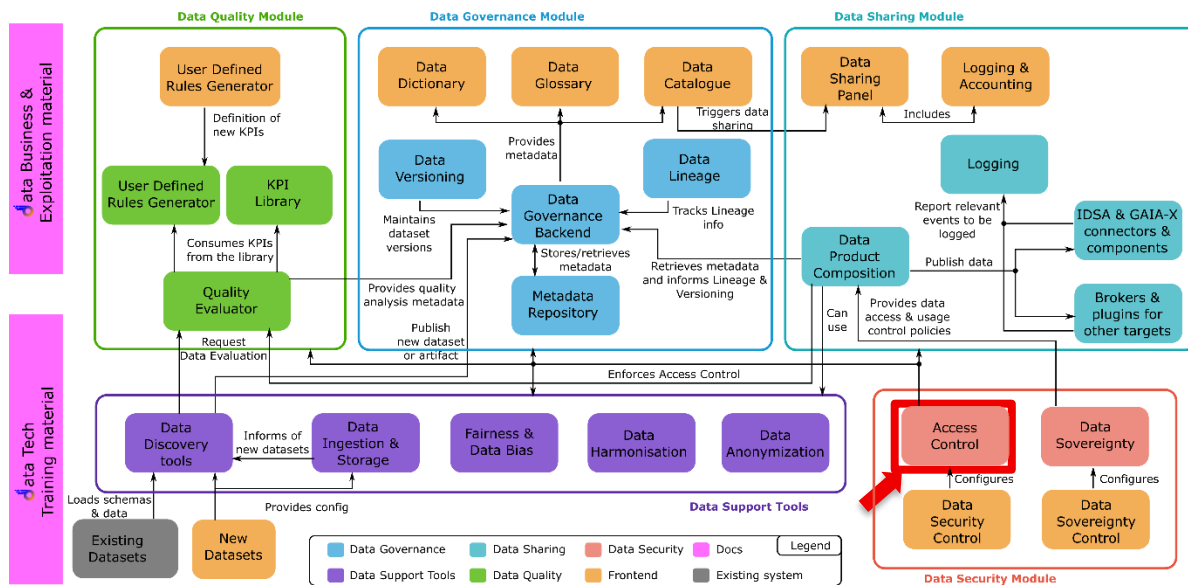


Figure 56: The Access Control component in the context of DATAMITE's architecture

Figure 57 illustrates the workflow of the Access Control within the DATAMITE framework. The Access Control component is a part of the security module and will be integrated into the frontend security panel. Users interact directly with the Access Control component for login and user account management. Moreover, the access control component communicates with other components to ensure fine-grained access control. Specifically, registered components redirect user requests to obtain the relevant token for validation, ensuring secure RBAC through this communication.

4.1.2 Status at M18

A Dockerfile was developed for a Docker image for Keycloak that used a Docker Compose file created to streamline the deployment process of the Keycloak-based access control system. This file defines the services, networks, and volumes required to run Keycloak and its dependencies in a containerised environment. By using Docker Compose, consistent and reproducible deployments across different environments can be ensured. To facilitate rapid development and testing, Keycloak was configured to run in development mode. This mode allows the user to make changes quickly and see their effects without the overhead of production-level configurations. In addition, PostgreSQL was integrated as the backend database for Keycloak. This database stores

all user information, configurations, and operational data. Finally, several pre-configurations were implemented. In more detail, an admin account was set up with the necessary privileges to manage the Keycloak server. This account is used for initial setup and configuration tasks. Moreover, a *realm* was pre-configured. Realms in Keycloak are isolated units of management that allow us to separate different clients, users, and configurations. Finally, a set of predefined roles was established within the realm. These roles assign specific permissions to users and groups, ensuring that access control policies are enforced consistently across the system. In more detail, the roles are described in Table 1. There is a [repository](https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-security/access-control)¹² in Eclipse for more information about the access control component.

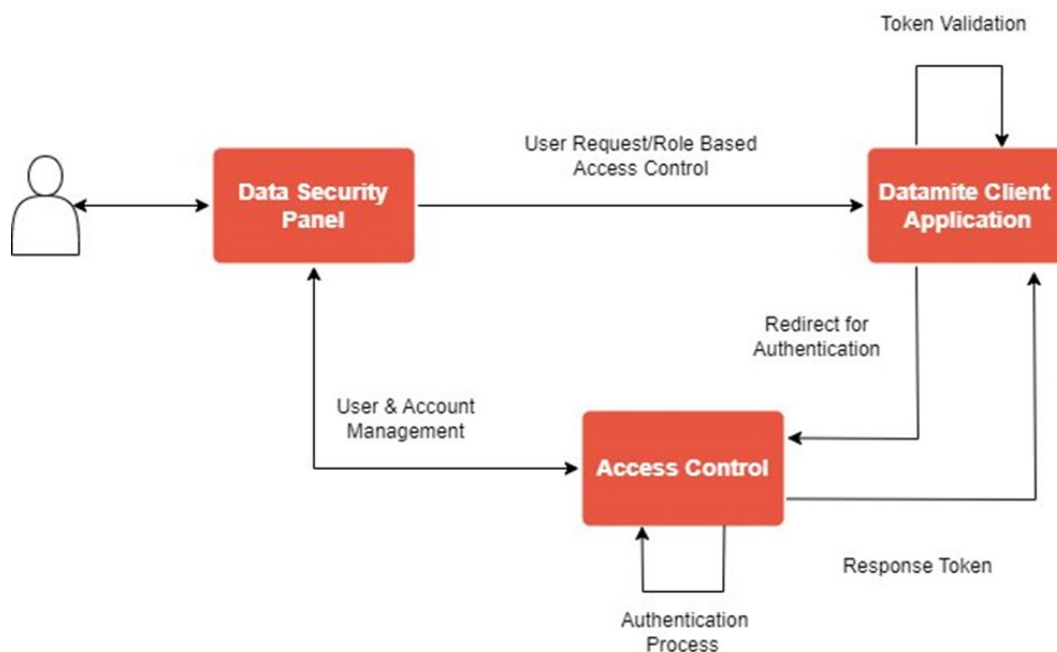


Figure 57: Data Access control workflow

¹² <https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-security/access-control>



Role Name	Description
Data Owner (Business User)	Responsible for datasets that are provided, authorising what datasets will be shared to whom and for how long
Data Consumer (Business User)	The user who will have access to data to use the data for business-related tasks and decision-making processes.
Data Provider	Responsible for preparing, making available, integrating and providing needed datasets/information to data consumers based on directions of data owners. Technical role that implements data owner directions
System Administrator	Responsible for the operation, management and maintenance of the installed DATAMITE instance
Security/Privacy Responsible	Responsible for setting the privacy and security policies, guiding/consulting users concerning related topics and overseeing their policy implementation.
Data Governance Responsible	Develops, consults, monitors and enforces data governance policies and practices (including Data Quality)

Table 1: DATAMITE Security User Roles



At this stage, the focus was on user management and the administrative activities related to managing users. The available functionalities included:

- Obtain an access token after successful authentication: Users can receive an access token upon successful authentication, enabling secure access to resources.
- Register a new user to an existing realm: Administrators can add new users to the existing realm, facilitating user onboarding.
- Update user information of an existing account: Administrators can modify the details of existing user accounts to ensure the accuracy and relevance of user data.
- Delete an existing user: Administrators can remove users from the system as needed.
- Query all registered users: Administrators can retrieve a list of all users currently registered in the system.
- Query a user by username/email/first name/last name: Administrators can search for specific users using their username, email, first name, or last name.
- Query user's representation: Administrators can obtain detailed information about a user's account and its attributes.

4.1.3 Final Status

This section describes the functionalities implemented in the final version of the Access Control component, which is responsible for user management within the DATAMITE framework, ensuring both authentication and role-based access control (RBAC). The defined roles for the DATAMITE framework are described in 4.1.2. In addition, the themes have been adapted based on DATAMITE's stylesheet (as shown in Figure 58 and Figure 59) to ensure consistency, adoption, and integration with the user interface.

For RBAC, a preconfigured realm has been created that includes all DATAMITE services registered as clients. For each client, the authorisation services have been enabled. Within these authorisation services, permissions (Read/Write/Delete) and the corresponding policies have been defined. These policies are paired with permissions and the realm's roles. Users are assigned one or more of these roles, ensuring that access to each service is granted or denied based on the assigned role(s).

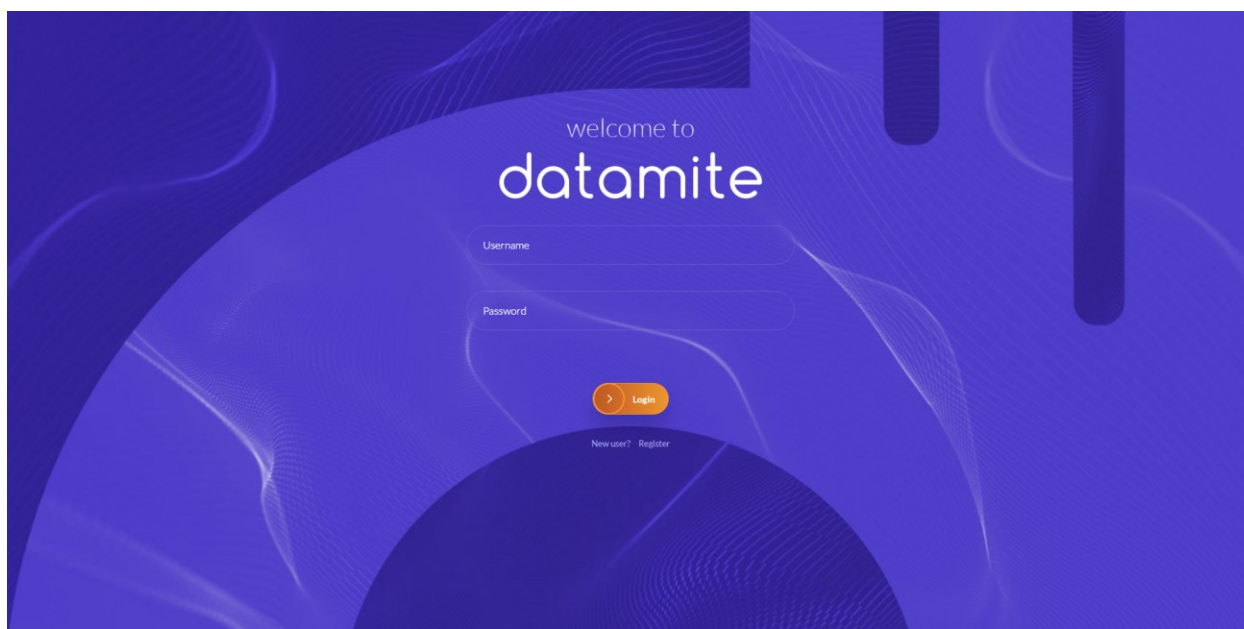


Figure 58: Access Control's login page

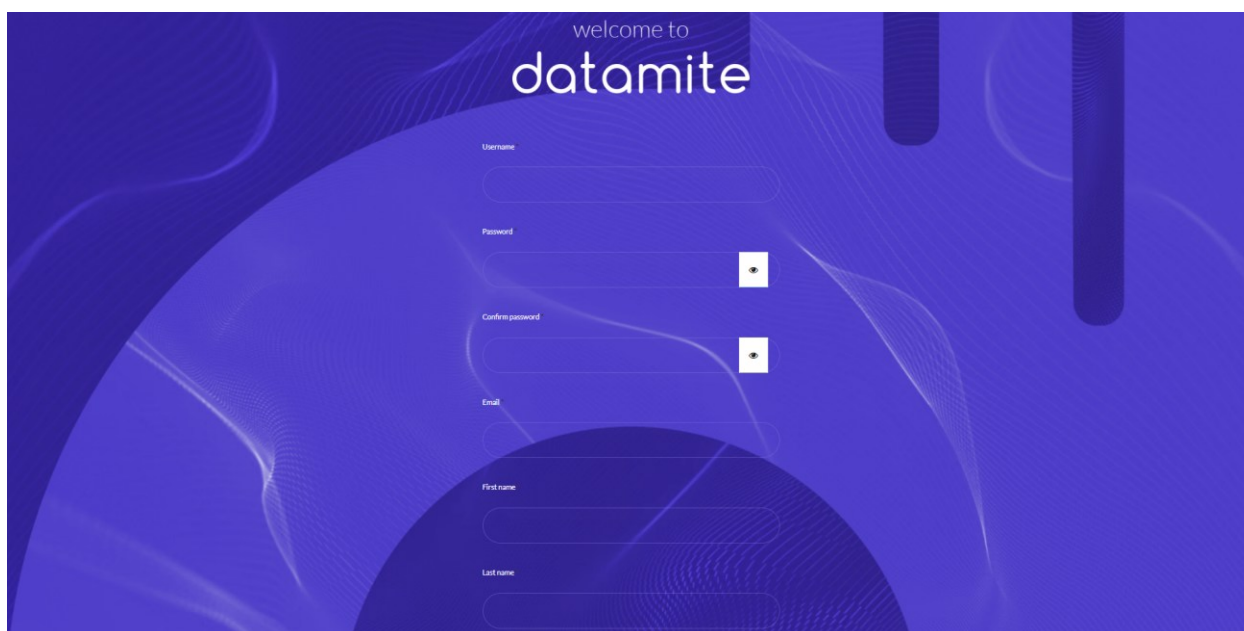


Figure 59: Access Control's registration page

From a technical perspective, when a user logs in to the Access Control component, upon successful authentication, the user is redirected to the main DATAMITE tab. At the same time,



Access Control forwards the user's token to the DATAMITE frontend. The frontend validates this token and, for each request, forwards it to the backend services. Each backend service then passes the token back to Access Control for validation. In response, Access Control returns a Boolean value (true/false), indicating whether the user is authorised and assigned with the appropriate role(s), based on the defined policies, to perform the requested action.

4.2 Data Sovereignty

The status of this component has already been described in *D2.2 Data Sharing and Sovereignty Mechanisms*. Please find the contents related to Data Sovereignty in Section 5 of D2.2.



5 Data Support Tools

This section presents the final version of the Data Support Tools, namely, the Data Ingestion & Storage, Data Discovery Tools, Fairness and Data Bias, and Data Anonymisation. This section presents their design, previous status and next steps. Note that Data Harmonisation is mentioned, but it is described in a different deliverable.

5.1 Data Ingestion and Storage

DATAMITE considers three types of data sources:

- The first two concern data stored in the framework, which can be uploaded in bulk or may come from streaming connections.
- The third one is external data from databases or repositories that the framework can access.

The Data Ingestion and Storage component focuses on the first two cases, bulk and streaming, in addition to the tools to store this data in the framework, which may be used as well by other components. Figure 60 shows the components within the context of the entire architecture.

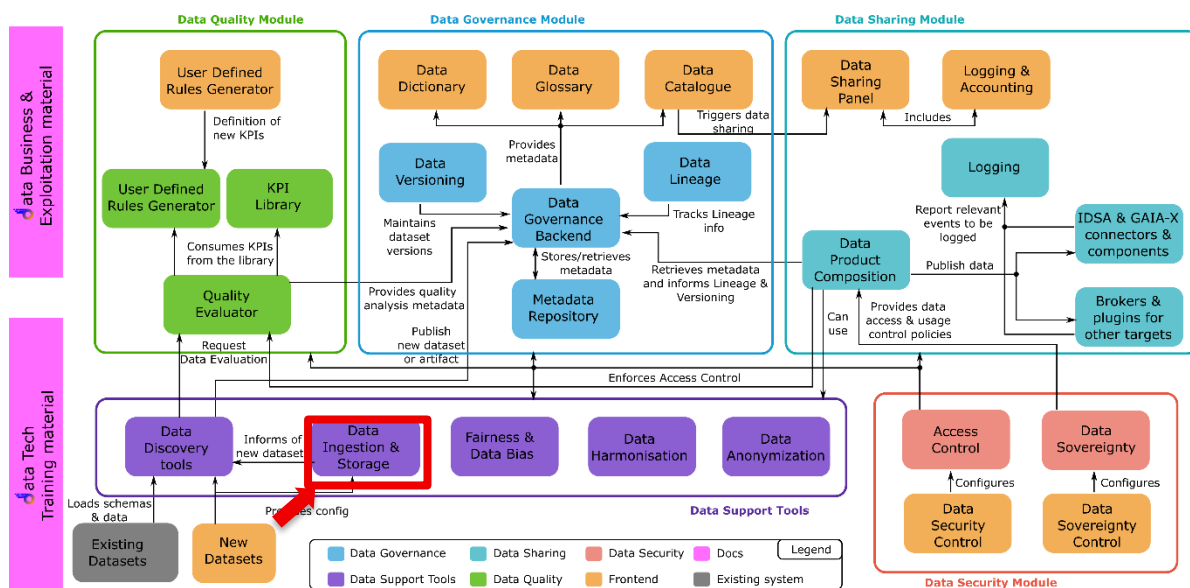


Figure 60: Data Ingestion and Storage in DATAMITE's architecture.



5.1.1 Design Overview

DATAMITE bulk ingestion is performed by leveraging the S3 Multipart library¹³, which allows data partitioning, among other features and can be used to upload large files conveniently. Regarding streaming data, DATAMITE adopts a plugin-based approach that allows for receiving data with different protocols (e.g., Kafka, MQTT, OPC-UA, etc.). Upon a new connection, it deploys a subscriber or client that connects to the data source and re-publishes the data into an internal Kafka cluster. Data received is stored temporarily in a landing layer until compacted according to time (e.g., every hour, every day, etc.) or volume (e.g., every 100MB, every 1GB, etc.) conditions. Every time data is compacted, it is passed to the persistence layer as a new data artifact. Data uploaded as bulk or received through streaming connections is stored in an internal MinIO instance. Similarly, once it is moved into the permanent storage, the Discovery Tools are invoked so that the new artifact is identified by the framework. An overview of the operational workflows within the component is shown in Figure 61.

5.1.2 Status at M18

The initial version of the component included the S3 library for bulk ingestion and it can receive streaming data with three protocols: Kafka, MQTT and OPC-UA. In the Kafka and MQTT cases, the framework deploys a subscriber to connect to an existing broker. Note, however, that OPC-UA is not a publish/subscribe protocol but client/server. In this case, DATAMITE deploys a client that will connect to the OPC-UA server to pull the data.

The deployment of the Kafka cluster, subscribers and internal data sinks is performed using Docker and a script-based implementation. Similarly, storage with MinIO was already possible, being both layers, landing and persistent, already operative. Current efforts are related to the integration with other components.

¹³ <https://docs.aws.amazon.com/AmazonS3/latest/userguide/mpuoverview.html>



More detailed and updated information can be found in the Eclipse Repo for the [Data Ingestion and Storage component](#)¹⁴.

5.1.3 Final Status

This component has evolved substantially both in functionality and complexity. First, the number of potential data source types has increased from three to four. During the development of the Data Product Composition component, it was considered that it may be necessary to query databases for particular sets of data. This gained relevance and evolved until it was considered necessary to facilitate this type of data ingestion, which needed its own metadata structures and to be considered as a new type of data artefact. This kind of ingestion, denoted as DB Queries, shares with databases the connections that have already been defined and uses Trino to execute such queries. Trino brings in an additional feature, as it allows the execution of queries to multiple databases using different database technologies (e.g., MySQL and MongoDB) at the same time. In addition, as an extra functionality, DB Queries can be extended to allow recursion on the queries being executed, that is, allow the execution of a query periodically during a period of time (e.g., every 12 hours for 5 days).

Secondly, DB Queries and the rest of the ingestion mechanisms have been migrated to Mage.ai, which allows for the automated creation and management of pipelines. This integration spans from the ingestion to the interaction with the data quality and data governance modules, as well as managing the storage of data in the local MinIO.

Regarding the streaming ingestion, the number of protocols accepted has not been extended, being compatible with Kafka, MQTT and OPC-UA, as pilots did not have further needs. However, the plugin-based approach followed facilitates the implementation of additional extensions for other protocols in case a user or organisation needs to connect with a different kind of data source. The final architecture of the component can be seen in Figure 61.

¹⁴ https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-support-tools/data_ingestion_and_storage

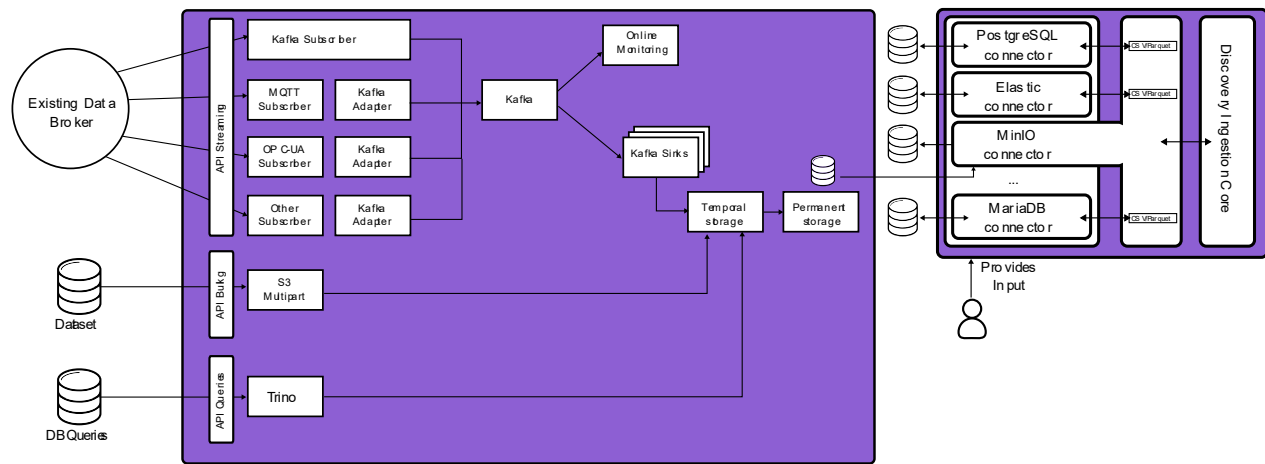


Figure 61: Data Ingestion and Storage workflows.

Finally, in order to facilitate the publication of datasets to data spaces or initiatives where data remains in DATAMITE and it is accessed remotely, such as in data spaces, the Data Storage API has been extended to indicate whether a data asset is expected to be consumed internally or externally -as it may be the case with some data products- to store them separately and improve security. Also, in relation to security aspects, the component has integrated the interaction with the access control component, controlling the actions that can be performed by the different users depending on their roles.

5.2 Data Discovery tools

The goal of the Data Discovery tools is to facilitate the extraction of metadata from the different datasets being added to DATAMITE, regardless of whether they are stored in a remote storage repository or within the framework through the Data Ingestion and Storage component. Figure 62 shows the Data Discovery tools in the context of architecture, showing its connections with the Data Ingestion and Storage component, the Data Governance Backend, the Quality Evaluator, and the frontend for the addition of new datasets or those existing datasets in remote storage repositories.

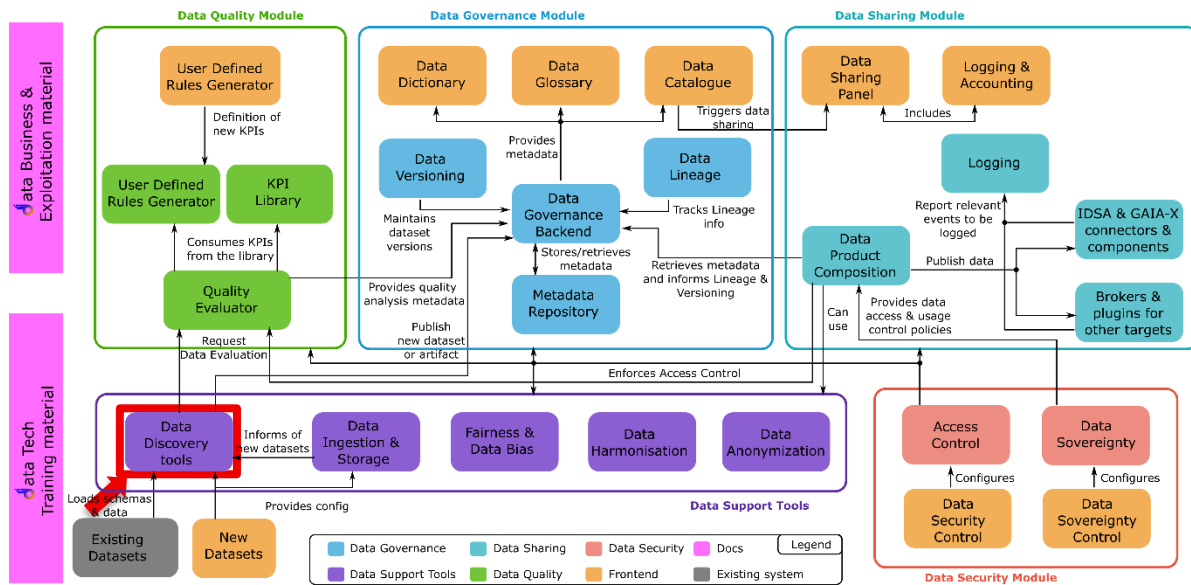


Figure 62: Data Discovery tools in DATAMITE's architecture.

5.2.1 Design Overview

The process to extract metadata from data is identical, regardless of whether the data is stored within DATAMITE or in an external repository. As in the case of streaming, DATAMITE follows a plugin-based approach with a collection of plugins to connect to different storage technologies, as can be seen in Figure 63. The plugins and core elements perform the following actions:

- Connect to the storage (e.g., a database) according to a configuration provided by the user.
- Extract the structural information from the data asset(s) requested and transform this information to the DATAMITE metadata schema.
- Publish the metadata object to the core element in the Data Discovery tools. The core element will push it to the Data Governance Backend.
- Back into the plugin, navigate the structural information extracted from the dataset to extract the data from the data source and create the profiling tasks to be executed by the Data Quality module to produce the initial set of quality metadata. Data is extracted on a column basis in principle.

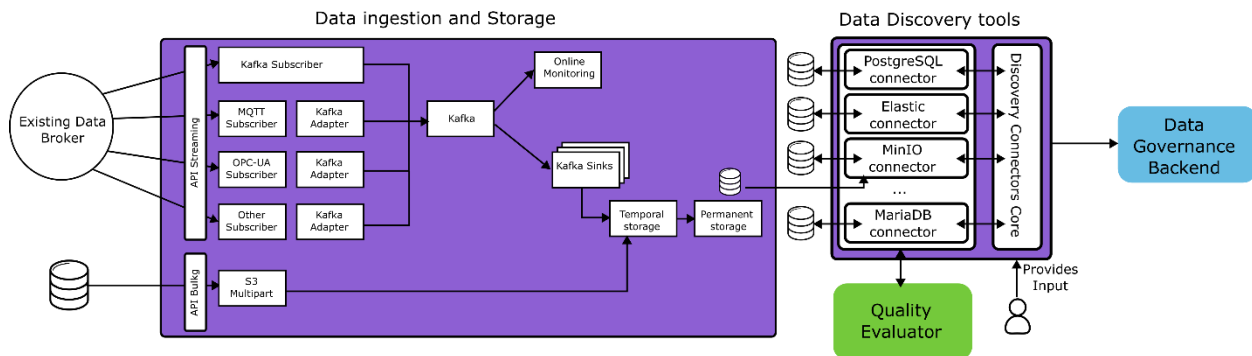


Figure 63: Data Ingestion and Storage relation to Data Discovery Tools

This is performed via two pipelines. The first one has two steps, comprising connecting to the storage and extracting the metadata, and secondly, publishing it. The second pipeline covers to perform the profiling of data. The information to perform the connection to the storage repositories and the assets to collect is provided by the user through an interface. Finally, note that, as our internal storage is based on MinIO, the ingestion and storage module would be linked to the MinIO connector, as can be seen in Figure 63. The internal pipelines are depicted in Figure 64

Finally, the different pilots in DATAMITE were asked to report what storage technologies they use, so those were the first technologies to be compatible with DATAMITE and whose plugins were used within the project execution. Table 2 enumerates the technologies that were reported by the different pilots.

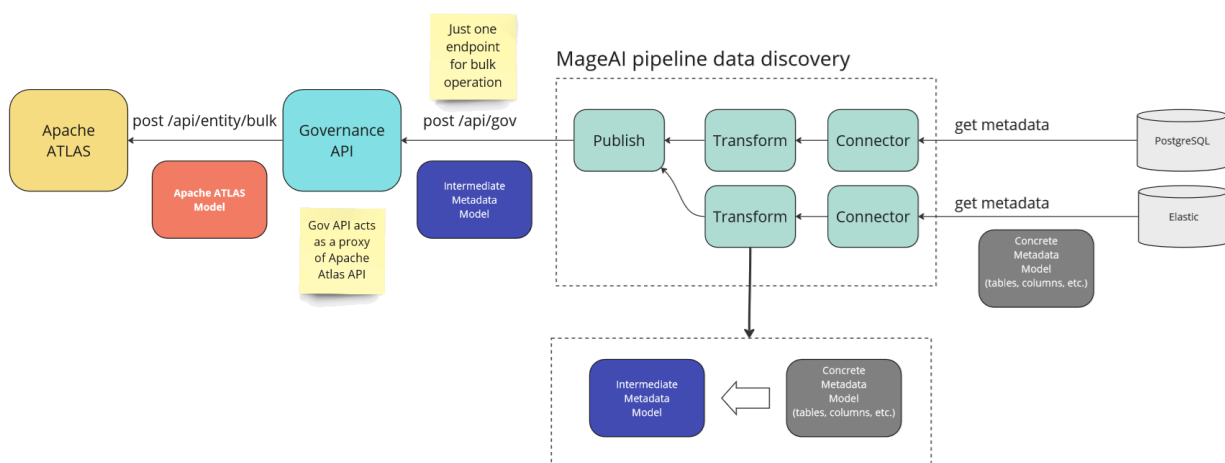


Figure 64: Data Ingestion and Storage pipelines



Database technology	Pilots using it
ElasticSearch	Pilot 1: Corporate Multi-Domain Data Exchange with EDIH support
PostgreSQL	Pilot 1: Corporate Multi-Domain Data Exchange with EDIH support Pilot 2- Corporate Multi-Site Data Exchange Pilot 3 - Offering Data to Service Providers with data spaces Pilot 6 - Connecting MISTRAL to the EU AIoD Platform
Cassandra	Pilot 1: Corporate Multi-Domain Data Exchange with EDIH support
S3 object storage (e.g., MinIO)	Pilot 2- Corporate Multi-Site Data Exchange Pilot 5 - Connecting eDWIN to Data Markets
MariaDB	Pilot 2- Corporate Multi-Site Data Exchange
MS SQL	Pilot 3 - Offering Data to Service Providers with data spaces
Azure Data Lake Storage Gen2	Pilot 4 - Leveraging Electricity Distribution Open Data
MongoDB	Pilot 1: Corporate Multi-Domain Data Exchange with EDIH support Pilot 5 - Connecting eDWIN to Data Markets
Kafka	Pilot 5 - Connecting eDWIN to Data Markets
Virtuoso	Pilot 5 - Connecting eDWIN to Data Markets
Redis	Pilot 6 - Connecting MISTRAL to the EU AIoD Platform

Table 2: Proposed technologies for compatibility with DATAMITE.

5.2.2 Status at M18

At M18, there was an initial deployment with Mage.ai implementing the workflow depicted in Figure 64. In addition, the plugins for PostgreSQL, Cassandra, Elastic and MinIO had already been implemented, while others, such as MongoDB and Virtuoso, were in development and



integration within Mage.ai. More detailed and updated information can be found in the Eclipse Repo for the [Data Discovery Tools](https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-support-tools/data-discovery-connectors)¹⁵.

5.2.3 Final Status

The development of the Data Discovery Tools has advanced in parallel with the Data Ingestion and Storage component in terms of its automation, its integration in Mage.ai, and some shared operational aspects. Besides the Mage.ai integration, the required metadata entities were defined and deployed in the metadata repository or the required frontend interfaces were developed.

Regarding the component itself, its main features are:

- The component has included connectors for Cassandra, PostgreSQL, MongoDB, MariaDB, Azure and MinIO. Although other connectors were considered initially, as described in Table 2, the storage technologies were eventually discarded by the pilots and converged to this set of technologies. The approach used is plugin-based, so it is easy to extend the functionality to other database technologies by adding new plugins.
- The creation of new database connections. It is relevant to mention that the user does not need to introduce information for a particular database connection every time. The definition of a new connection is stored and can be used. Moreover, the connection data is also used to create a Trino config file so this connection can be used, as well, by the DB Queries ingestion mechanism.
- The aforementioned integration with Mage.ai has been carried out, so when the discovery of a database is required, it triggers a pipeline that manages this process.
- The Data Discovery Tools have been integrated with the Access Control component to manage what the different users can do according to their role.

¹⁵ <https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-support-tools/data-discovery-connectors>

5.3 Fairness & Data Bias

The Fairness & Data Bias aims to assist DATAMITE users in assessing and potentially mitigating Bias present in their data when datasets are used in decision-making processes. Data bias fairness refers to the principle of ensuring that such datasets are representative and devoid of biases that could lead to unfair or discriminatory outcomes. It involves identifying, mitigating, and preventing biases in data collection, processing, and analysis to promote equitable treatment and outcomes for all individuals or groups affected by the decisions made using that data. This concept aligns with the broader framework of data quality and is a crucial part of DATAMITE's value proposition. The Fairness & Data Bias is one of the Data Support Tools in DATAMITE, as shown in Figure 65.

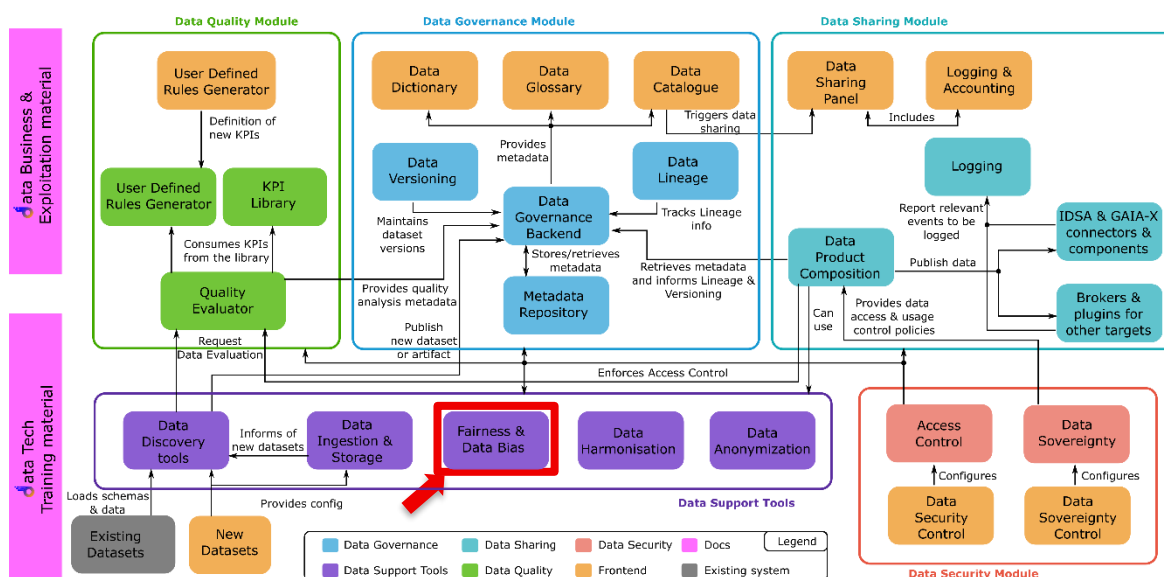


Figure 65: Fairness & Data Bias tool in DATAMITE.

5.3.1 Design Overview

Figure 66 describes the intended workflow for a user who seeks to assess the level of fairness bias in their dataset. Because fairness is contextual, we identified a workflow that accommodates a wide range of different fairness considerations in tabular data. The workflow is divided into two phases:

The Query phase, where the user formulates fairness queries designed to address specific fairness considerations relevant to their business needs.

And the Results phase, where the user can view the results of each query and generate a comprehensive fairness report.

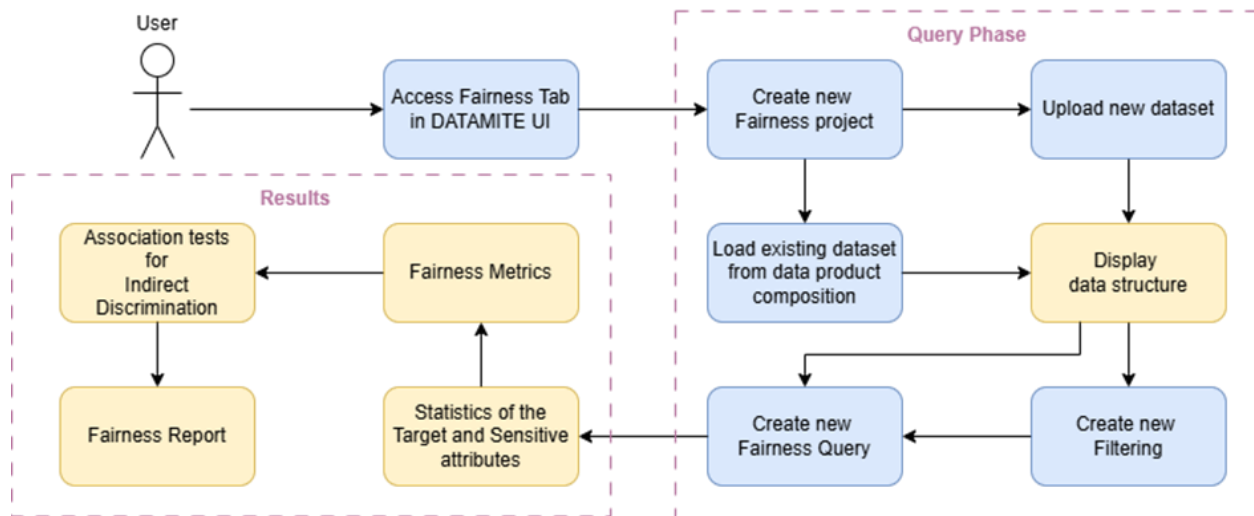


Figure 66: The workflow of the fairness assessment tool.

The full workflow includes the following steps:

- Through the DATAMITE interface, the user logs in to the system and accesses the Fairness & Data Bias tool tab.
- The user creates a new fairness Project in the dedicated MongoDB database by uploading their data to the system. Alternatively, the user can create a new project by loading an existing dataset from the data product composition service.
- Once a new project has been created, the tool presents the dataset structure, which contains important information for each column of the dataset, including data types, unique values, and a suggested data variable type. This enables the user to understand the dataset, identify potential inconsistencies in the data, or missing values denoted by a specific symbol, and more.



- The user may define filters to be applied prior to the analysis, such as columns to drop and specific values to retain. This allows the user to focus their analysis on a specific subset of the problem space or filter out irrelevant features (like identifiers or duplicated information) and irrelevant values.
- The user formulates the target attribute for the analysis. This is an important component of the fairness evaluation, as the user defines the outcome or the attribute(s) for which decisions are made, and which could potentially lead to discrimination. Some indicative target attributes include income, loan eligibility, being hired, and others.
- The user then defines the sensitive or protected attribute(s) to which the decision-making process should not be attached. This is a key component of the analysis, as it involves attribute(s) that commonly experience discrimination in society. Common examples of sensitive attributes are Gender, Ethnicity, Religion, Race, Age, and more.
- The user may also select attributes that are contextually associated with the sensitive attribute and could therefore contribute to discrimination. These attributes potentially share information with the predefined sensitive attribute(s). This enables the user to assess for indirect discrimination, which refers to bias perpetuated through strong proxy features for the sensitive attribute. If such attributes exist, simply excluding the sensitive attribute will not resolve the fairness problem, as these proxy attributes still carry information about the sensitive attribute. Common proxy features for gender (when it serves as a sensitive attribute) include height, weight, and more.
- The user may also select the classifier that is intended to predict the predefined target attribute. This is another important component that measures to what extent potential discrepancies are retained or amplified by a machine learning model.
- Once everything is set up, the user can run a query.

After the results are computed, they are made available to the user. The fairness results include:

- Important descriptive statistics for the sensitive and target attributes.

- The most important fairness metrics from the literature, including demographic parity, disparate impact, equal opportunity and equalized odds for both the actual labels of the dataset and the predictions derived by the model.
- Confusion matrices for two approaches – a classifier trained with the sensitive attribute as part of the dataset, and a classifier trained without the sensitive attribute.
- Some statistical tests to measure the strength of potential associations between the sensitive attribute(s) and the selected contextually associated features.

The user selects the queries whose results will be included in the fairness report for PDF export.

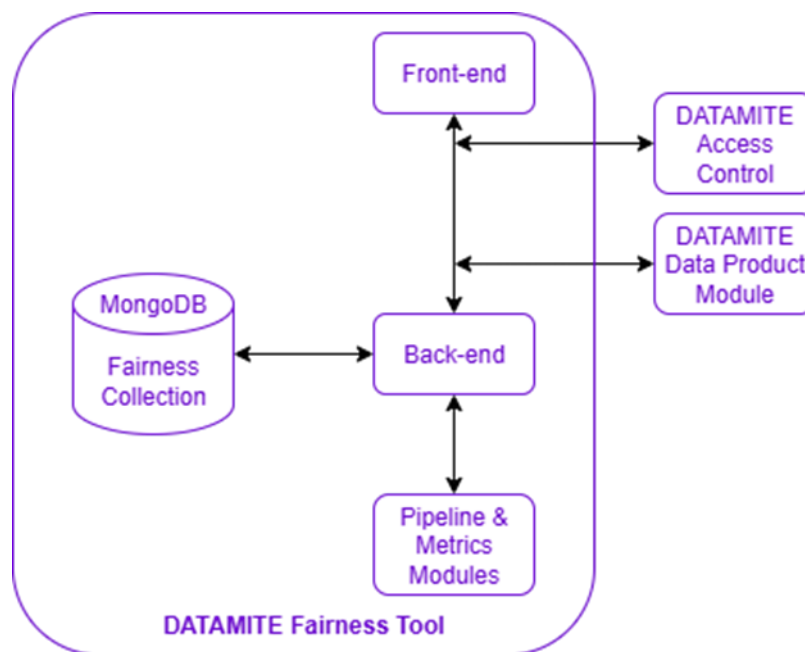


Figure 67: The architecture of the fairness assessment tool.

The proposed architecture for the Fairness and Data Bias tool, enhancing the workflow described above, can be found in Figure 67. The module communicates with other DATAMITE components, such as the Access Control module for login, the DATAMITE's MongoDB database that contains a fairness collection with the created JSON configuration files for each fairness project, and the Data Product composition module from which the user can input a dataset to the fairness tool and attach the fairness report to it.

The tool components are:

- **Backend:** A Python Flask web service that manages the communication between the module and other DATAMITE components for user authentication, and data ingestion. It also manages the functionalities provided by the Fairness components, and the communication with the MongoDB database. Figure 68 and Figure 69 present the endpoints that manage the aforementioned functionalities that allow the user to complete an end-to-end fairness analysis.
- **Frontend:** An intuitive and user-friendly interface that enables users to create fairness projects, formulate fairness queries, view results and generate/attach fairness reports to the data product.
- **Database:** A dedicated fairness collection within a MongoDB database to store, retrieve, and update the JSON configuration documents for each created project.
- **Pipeline Module:** Provides a set of tools useful for data preparation and the execution of fairness queries.
- **Metrics Module:** Contains a compendium of the most relevant Fairness metrics available in the literature for data bias assessment.

Project Operations related to projects.			^
POST	/project/add_project	Add Project API Endpoint	⌵ 🔒
POST	/project/add_project_with_id		⌵ 🔒
DELETE	/project/delete_project/{id}	Delete Project by Id API Endpoint	⌵ 🔒
GET	/project/get_data_structure/{id}	Get Data Structure by Id API Endpoint	⌵ 🔒
GET	/project/get_project/{id}	Get Project by Id API Endpoint	⌵ 🔒
GET	/project/get_projects	Get all Projects API Endpoint	⌵ 🔒
PUT	/project/update_project/{id}	Update Project by Id API Endpoint	⌵ 🔒 📄
Transformation Operations related to transformations.			^
POST	/transformation/create_transformation/{project_id}	Create Transformation API Endpoint	⌵ 🔒
DELETE	/transformation/delete_transformation/{project_id}/{transformation_id}	Delete Transformation by Id API Endpoint	⌵ 🔒
PUT	/transformation/edit_transformation/{project_id}/{transformation_id}	Edit Transformation by Project and Query Id API Endpoint	⌵ 🔒
GET	/transformation/get_transformation/{project_id}/{transformation_id}	Get Transformation by Project and Transformation Id API Endpoint	⌵ 🔒
GET	/transformation/get_transformations/{project_id}	Get Transformations by Project Id API Endpoint	⌵ 🔒

Figure 68: Endpoints to create a new fairness project and filtering



Query		Operations related to queries.	^
POST	/query/create_query/{project_id}	Create Query API Endpoint	✓
DELETE	/query/delete_query/{project_id}/{query_id}	Delete Query by id API Endpoint	✓
GET	/query/get_queries/{project_id}	Get Queries by Project id API Endpoint	✓
GET	/query/get_query/{project_id}/{query_id}	Create Query by Project and Query id API Endpoint	✓
Fairness Analysis		Operations related to Fairness.	^
DELETE	/fairness/delete_result/{project_id}/{result_id}	Delete Result by Project and Result id API Endpoint	✓
POST	/fairness/fairness_analysis_report/{project_id}		✓
GET	/fairness/get_result/{project_id}/{query_id}	Get Result by Project and Result id API Endpoint	✓
GET	/fairness/get_results/{project_id}	Get Result by Project and Result id API Endpoint	✓
POST	/fairness/start/{project_id}/{query_id}	Generate Result by Project and Query id API Endpoint	✓
POST	/fairness/stop/{project_id}/{query_id}		✓

Figure 69: Endpoints to create and run fairness queries.

5.3.2 Status at M18

This version of the prototype contained advanced versions of the backend, Descriptive Statistics and Metrics Modules, allowing for a full iteration of the Detection phase of the workflow. Some of the functionalities that were already available are:

- Direct upload of .csv and .xlsx files for analysis in the absence of a more integrated data ingestion mechanism.
- APIs for creation, modification and deletion of a Project object storing experiment metadata and results in the dedicated database.
- Several descriptive statistics functionalities for the characterisation of the dataset: automatic detection of the type of data, pre-processing encoding (ordinal, hot-one), measurement of proportions, etc., with their corresponding API endpoints.
- Relevant fairness metrics for bias detection that are independent of a predictive system: statistical parity difference and disparate impact, with their corresponding endpoints.

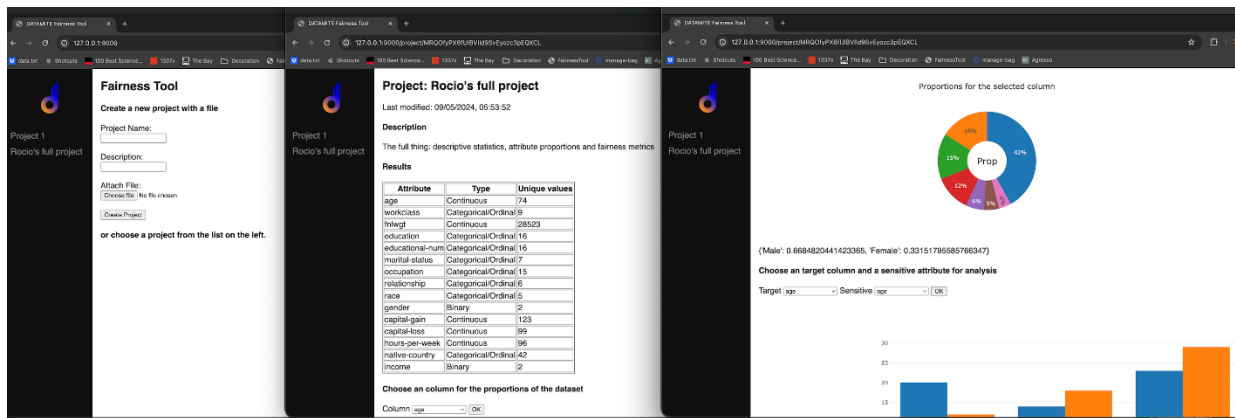


Figure 70: Prototype UI of the Fairness and Data Bias tool

For the testing of the endpoints and fairness functionalities, both Swagger documentation (Figure 68 and Figure 69) and a temporary user interface were created (Figure 70). Such a dual display aims to provide a testing playground for users who are either developers or non-technical.

5.3.3 Final Status

Over the last months, the focus of the efforts was on improving and adapting the workflow of the tool to cover a wide range of fairness considerations. Examples would be to identify the minimum inputs required from the user to conduct a complete fairness analysis and identify discrimination biases in tabular data, tailored to specific business needs, use cases or legal frameworks. These inputs include:

- Filtering applied prior to the analysis allows the user to drop irrelevant columns and filter out noisy values within the dataset.
- The target attribute can be a single attribute within the dataset or a combination of them. The user can formulate the target attribute by specifying: 1) the positive outcome and 2) whether to conduct a binary analysis. If a binary analysis is selected, C+ represents the selected favourable values and C- represents all remaining values. Alternatively, the resulting target attribute will include all possible combinations.
- The sensitive attribute can also be a single attribute or a combination of them. A similar functionality allows the user to define the sensitive attribute by specifying: 1) the privileged group (denoted as S+) and 2) the protected group (denoted as S-). If the protected group

is not defined, all possible combinations of the remaining values different to the privileged group will form the protected groups.

- Features for indirect discrimination are contextually sharing information with the sensitive attribute. Depending on the type of each selected feature, different association tests and statistical measures have been included, such as Mutual Information, Chi-square of independence (along with the Cramer's V measure), One-Way ANOVA (along with the effect size, Bonferroni and Tukey HSD for post-hoc analysis), Point Biserial Correlation and Kendall's Tau.
- The Classifier to predict the target attribute. The user can choose one of the integrated Machine Learning models, including Random Forest, Decision Tree, Logistic Regression, Naïve Bayes classifier, K-Nearest Neighbours and Support Vector Machine (SVM), along with their hyperparameters.

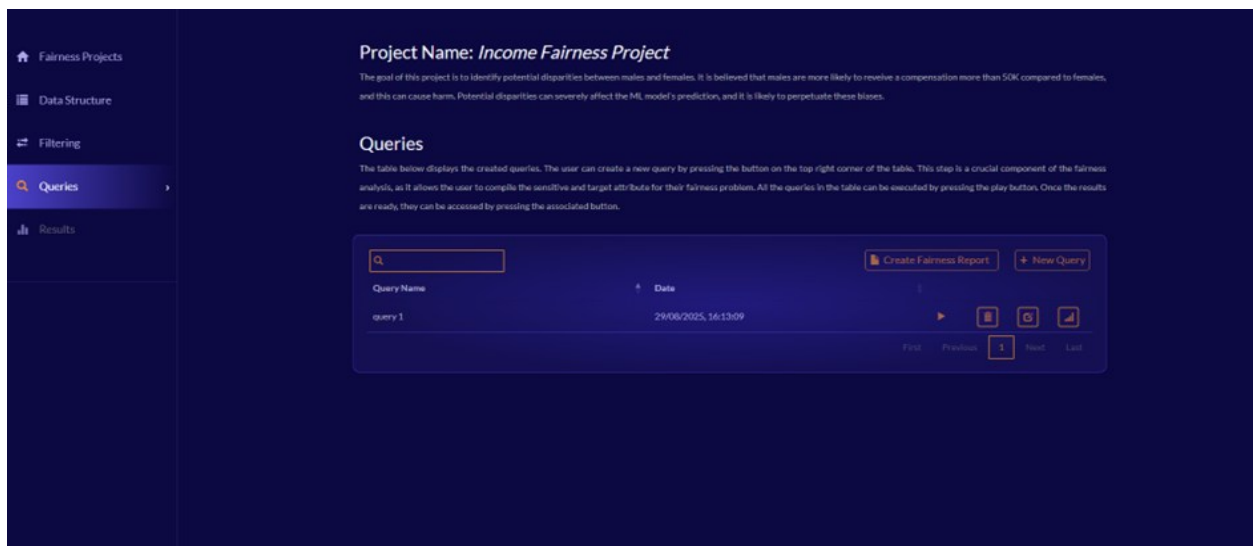


Figure 71: Queries tab of the fairness assessment tool

In addition to the tool's workflow, we focused on UI development. Figure 71 presents our interactive and intuitive UI of the fairness tool through which the user can create new fairness projects, filters and queries, which have been developed to streamline this complex but comprehensive fairness pipeline. With our fully integrated UI, the user can additionally run a query, view the results, and generate a fairness report in a PDF format with selected results. This

PDF report can be attached to the data product to better advertise and disseminate it. Figure 72 depicts the functionality that allows the user to create new filtering by inserting columns to drop and numerical ranges that match the desired filters. Figure 73 demonstrates the resulting equal opportunity metric for different measures within the Results tab.


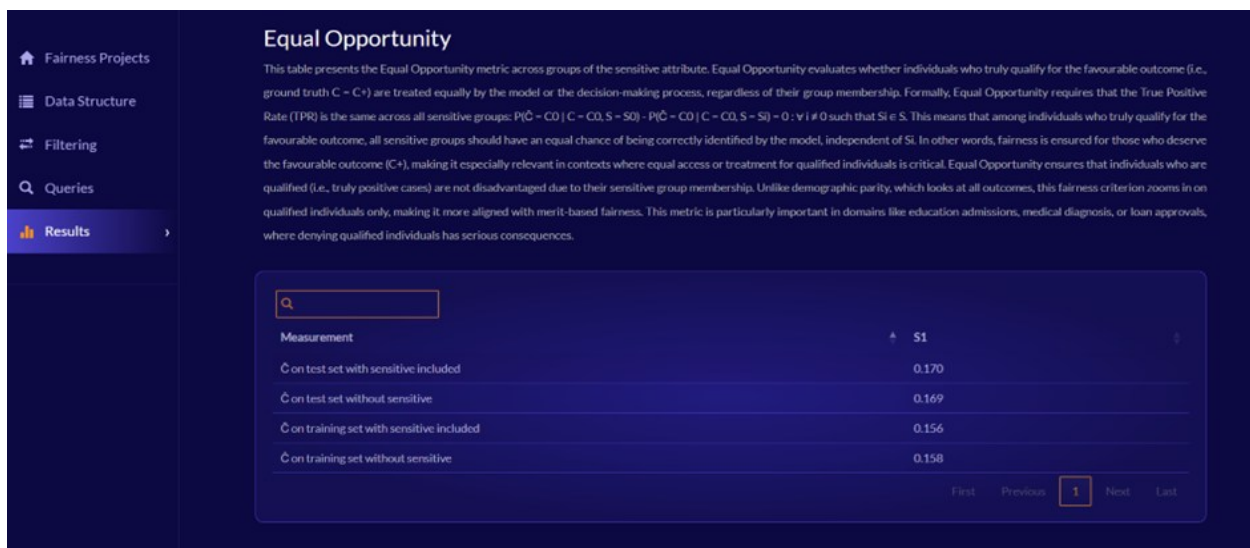


Figure 72: Indicative inputs to create new filtering.



Equal Opportunity

This table presents the Equal Opportunity metric across groups of the sensitive attribute. Equal Opportunity evaluates whether individuals who truly qualify for the favourable outcome (i.e., ground truth $C = C^+$) are treated equally by the model or the decision-making process, regardless of their group membership. Formally, Equal Opportunity requires that the True Positive Rate (TPR) is the same across all sensitive groups: $P(\hat{C} = C^+ | C = C^+, S = S_0) - P(\hat{C} = C^+ | C = C^+, S = S_1) = 0 : \forall i \neq 0$ such that $S_i \in S$. This means that among individuals who truly qualify for the favourable outcome, all sensitive groups should have an equal chance of being correctly identified by the model, independent of S_i . In other words, fairness is ensured for those who deserve the favourable outcome (C^+), making it especially relevant in contexts where equal access or treatment for qualified individuals is critical. Equal Opportunity ensures that individuals who are qualified (i.e., truly positive cases) are not disadvantaged due to their sensitive group membership. Unlike demographic parity, which looks at all outcomes, this fairness criterion zooms in on qualified individuals only, making it more aligned with merit-based fairness. This metric is particularly important in domains like education admissions, medical diagnosis, or loan approvals, where denying qualified individuals has serious consequences.

Measurement	S1
\hat{C} on test set with sensitive included	0.170
\hat{C} on test set without sensitive	0.169
\hat{C} on training set with sensitive included	0.156
\hat{C} on training set without sensitive	0.158

First Previous 1 Next Last

Figure 73: Equal opportunity metric in the Results view page.



In addition to the UI development, we integrated with the different DATAMITE components, including the Access Control for user login, which grants authenticated users access to our tool. Depending on their permissions, users can perform different operations. Write access allows users to create fairness projects, queries, and filters, as well as run queries. Read access enables users to view existing fairness projects, filters, queries and results. Finally, delete access permits users to remove fairness projects from the MongoDB database.

We have also integrated the fairness tool with DATAMITE's MongoDB database, where we store the JSON configuration files for fairness projects in a dedicated fairness collection. Users, based on their permissions, can push a document containing a fairness project's configurations. Once the document is in the database, they can create new filters and queries that update the project's document by adding relevant information. Once a query has been executed and results are computed, they are also added to the project's document within the fairness collection.

The last component that our tool has been integrated with is the Data Product Composition Service, which enables users to create a new project from an existing dataset. The user has the option to select the dataset they want to conduct fairness analysis for. To achieve this, we communicate with the Data Product Composition API to fetch the dataset and create a new fairness project.

Lastly, we identified that mitigation techniques should not be a part of this tool and should only focus on bias detection, transparency, and reporting. This choice was guided by several practical and ethical considerations. Firstly, many mitigation algorithms are highly contextual and applying them can risk overcorrecting or even introducing new forms of bias. Additionally, the tool is designed to be model-agnostic, making it difficult to enforce one-size-fits-all fairness corrections across different use cases. We believe that domain experts are best positioned to make informed decisions about when and how to mitigate bias, given the legal, cultural, and business implications involved. Moreover, some in-processing and post-processing techniques introduce significant



computational and implementation complexity, which did not align with the performance and usability goals of the tool. The tool is available in the dedicated [GitLab repository](https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-support-tools/fairness-data-bias)¹⁶.

5.4 Data Harmonisation

The status of this component has already been described in *D2.2: Data Sharing and Sovereignty Mechanisms*. Please find the contents related to Data Harmonisation in Section 4.3.

5.5 Data Anonymisation

The Anonymisation tool is responsible for protecting the sensitive information of an imported dataset while maintaining data utility. Figure 74 depicts the anonymisation tool within the context of DATAMITE's architecture. In more detail, the Anonymisation tool interacts with the Data Sharing Module and more specifically with the Data Product Composition, ensuring anonymity also for the data product. This chapter will provide a comprehensive overview of the design, architecture, workflow, development status, applied technologies, encountered issues/obstacles and future roadmap. Firstly, detailed explanations of the design and architecture, outlining the workflow of the Anonymisation tool, will be included. Furthermore, the status of development will be described, including functionalities and Swagger implementation. Any issues or obstacles encountered during the development of the Anonymisation tool will be specified. Finally, a roadmap for future actions and objectives will be provided, outlining the expected functionalities and planned development steps.

¹⁶ <https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-support-tools/fairness-data-bias>

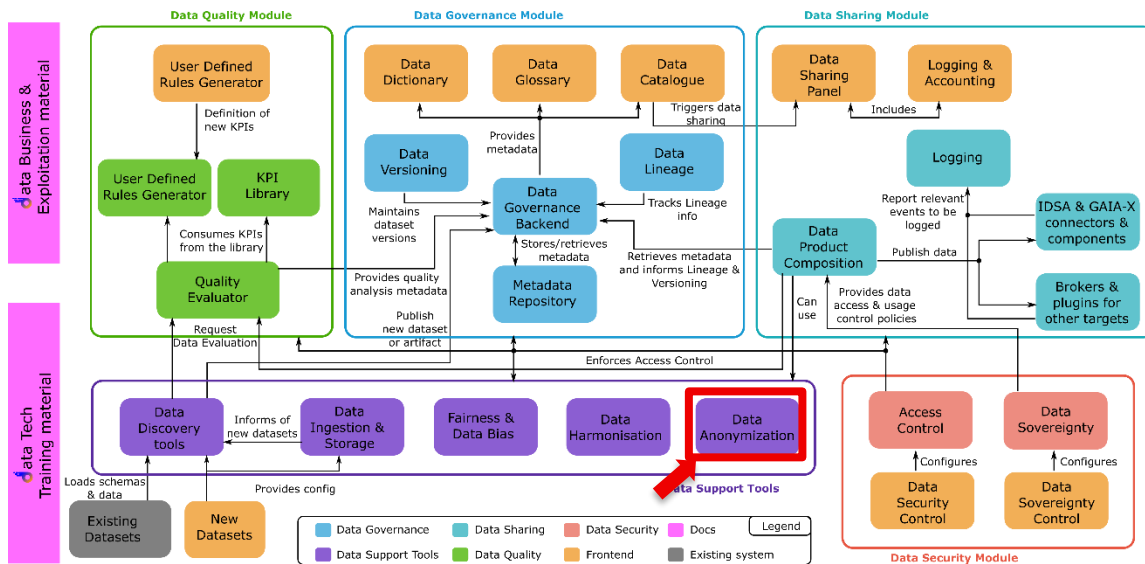


Figure 74: The Anonymisation Tool in the context of DATAMITE's architecture

5.5.1 Design Overview

This section describes the process required for the user to follow to anonymise a dataset. Figure 75 depicts the architecture of the anonymisation tool by briefly explaining how the tool works.

By using the anonymisation tab of the DATAMITE framework, the user has the option to import a dataset. The supported fields of the imported dataset are the general sensitive ones (email, phone number, name, age, etc.). Then, the Anonymisation tool will recognise the sensitive columns from the imported dataset, and the user will be able to select the desired columns, as well as the anonymisation technique for each column. The anonymised dataset will be stored in a database. Finally, the user will be able to retrieve the dataset and check if the dataset satisfies the k-anonymity property with k equal to two.

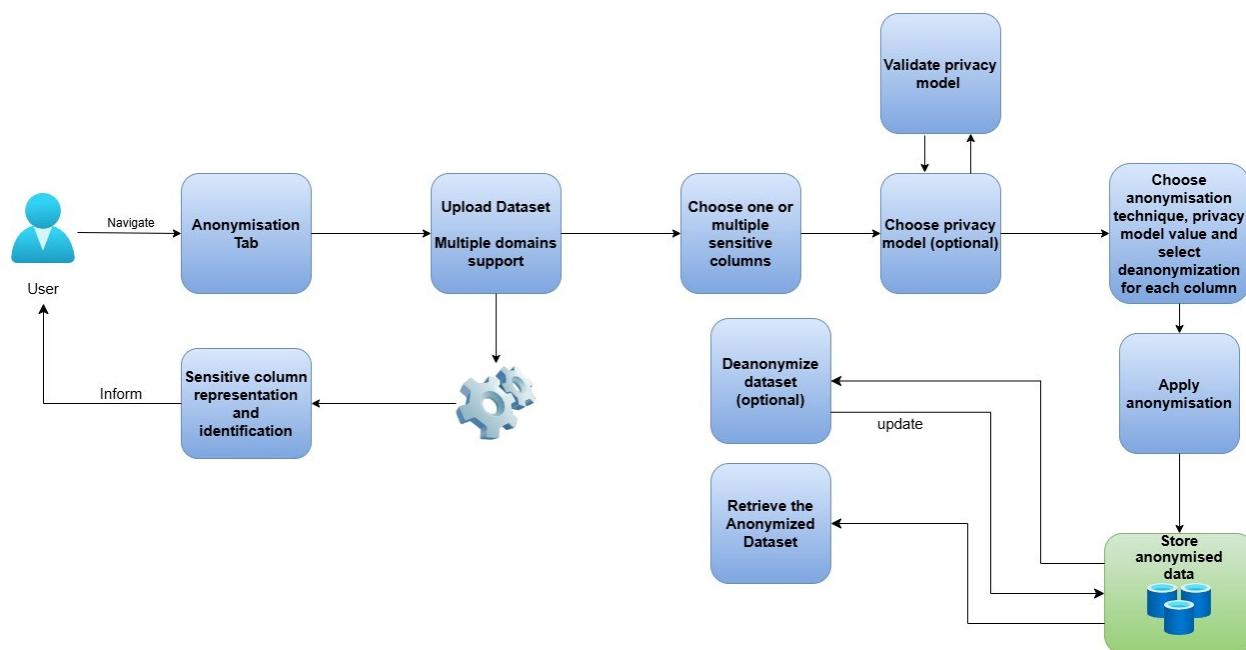


Figure 75: Anonymisation tool workflow

5.5.2 Status at M18

The Anonymisation tool is written in the Python programming language. Several Python libraries have been used in the implementation of most functions, such as nltk library¹⁷, pandas library¹⁸, fuzzy-wuzzy library¹⁹, etc. Similarly, swagger documentation has been developed to represent the endpoints of the Anonymisation tool to simplify the integration with the frontend. The corresponding Eclipse [repository](https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-support-tools/anonymization-tool)²⁰ contains more information about the Anonymisation tool.

The status of the developments of the Anonymisation tool at M18 included:

- The Anonymisation tool is integrated with the frontend environment of DATAMITE.

¹⁷ <https://www.nltk.org/>

¹⁸ <https://pandas.pydata.org/>

¹⁹ <https://pypi.org/project/fuzzywuzzy/>

²⁰ <https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-support-tools/anonymization-tool>



- The anonymised dataset is stored in a database.
- The tool supports a deanonymization process where the user can revert to the original dataset.
- The tool also supports other formats, e.g., JSON, in addition to CSV and XML formats for the imported dataset.
- The user can choose in which format they want to retrieve the anonymised dataset.
- The tool supports multiple domains through an anonymisation feature called sensitive lists, where the user can create their own sensitive columns
- The user will be able to insert, update and delete custom sensitive columns: The user will be able to insert their preferred columns for anonymisation in the anonymisation page of DATAMITE's framework. Those columns will be matched with columns of the imported dataset, and based on the type of data in the column, the appropriate anonymisation techniques will be presented to him to choose one of them.
- The tool supports privacy models like k-anonymity, l-diversity and differential privacy, and the user can choose one of them for each anonymisation process. Also, the tool informs the user if the privacy model is appropriate for the columns of the anonymisation process.
- The tool supports anonymisation and deanonymization of data products.

Figure 76 depicts the Swagger documentation for the Anonymisation tool. Swagger documentation is a frontend environment where the user or the developer will be able to test all the endpoints of the tool to see if the backend interacts well with the front end. The Swagger documentation consists of GET and POST requests, depending on the action we want to perform.



Load data Everything about load process and sensitive column identification		^
POST	/load_data Upload a csv,xlsx or json file	▼
GET	/get_data_product Get data product by unique ID or file name	▼
GET	/get_saved_data_fromanonymization Fetch anonymized data	▼
GET	/get_all_metadata Retrieve all metadata	▼
Custom Lists Process of custom list operations (add, update, delete)		^
GET	/get_predifined_sensitive_columns_by_datamite Get Supported Sensitive Columns	▼
POST	/add_custom_list_sensitive_columns Add Sensitive Columns in a custom list	▼
PUT	/update_custom_list_sensitive_columns Update Custom List	▼
DELETE	/delete_custom_lists Delete custom lists	▼
GET	/get_all_custom_lists Get All Custom Lists	▼
GET	/get_custom_lists Retrieve specific custom lists	▼
Identification Process Everything about column identification		^
POST	/identify_columns Identify Sensitive Columns	▼
GET	/get_sensitive_columns Retrieve sensitive columns data	▼
Anonymization process Process of the anonymization		^
POST	/post_validate_columns Validation of the selected column provided from the user	▼
POST	/post_column_info Sensitive column selection	▼
POST	/post_anonymization_info Anonymization technique selection for every column followed by anonymization process and database storing	▼
Deanonymization process Process of the deanonymization		^
PUT	/update_deanonymized_data Update anonymized dataset with original values	▼
GET	/get_deanonymization_columns Available columns that can be deanonymize	▼
POST	/post_column_to_deanonymize User can post specific anonymized columns to deanonymize	▼

Figure 76: SWAGGER Documentation for the Anonymisation tool

5.5.3 Final Status

The final developments of the Anonymisation tool include:

- The integration of the Anonymisation tool with DATAMITE's frontend.
- The storage of the anonymised dataset.



- Support for additional formats, e.g., JSON, in addition to CSV and XML formats for the imported dataset.
- The user can choose in which format they want to retrieve the anonymised dataset.
- The tool supports domain-specific sensitive columns such as meteorological, agriculture, telecommunication and energy.
- The user can insert custom sensitive columns: This enables the user to insert their preferred columns for anonymisation in the anonymisation page of DATAMITE's framework. Those columns will be matched with columns of the imported dataset, and based on the type of data in the column, the appropriate anonymisation techniques are presented to the user so she can choose one of them.
- Check data anonymisation: After anonymisation is finished, the user can check if the anonymisation satisfies the k-anonymity²¹ property with k equal to two. This means that each anonymised record from each anonymised column appears at least two times.

5.6 Data sampling recommender

The DATAMITE project faces a challenge with profiling large datasets that strain resources and take considerable time to analyse. A preliminary solution involves simplifying the dataset by using basic random sampling for analysis. However, the results from this reduced dataset cannot always be considered representative of the original data. Different sampling methods can impact the quality and relevance of the outcomes.

This section presents a sampling recommender prototype that recommends the most suitable data sampling technique/s based on the specific characteristics of the scenario. The recommender is implemented with a Decision Tree Classifier. The system follows a two-step process:

²¹ <https://academic.oup.com/jamia/article/15/5/627/732733>

- **Training:** The first step consists of learning with the provided training data. The obtained model during the training phase is stored as a binary object in a file to be loaded and used during the prediction step.
- **Prediction:** The second step consists of recommending the most appropriate sampling method/s, according to the provided input data and using the model obtained in the first step.

The decision tree is a classification algorithm which is implemented as a flowchart-like tree where the nodes represent the features, the branch represents a decision rule, and the leaf nodes represent the recommended algorithms. The following figure presents the tree that reflects the model generated for recommending the sampling methods:

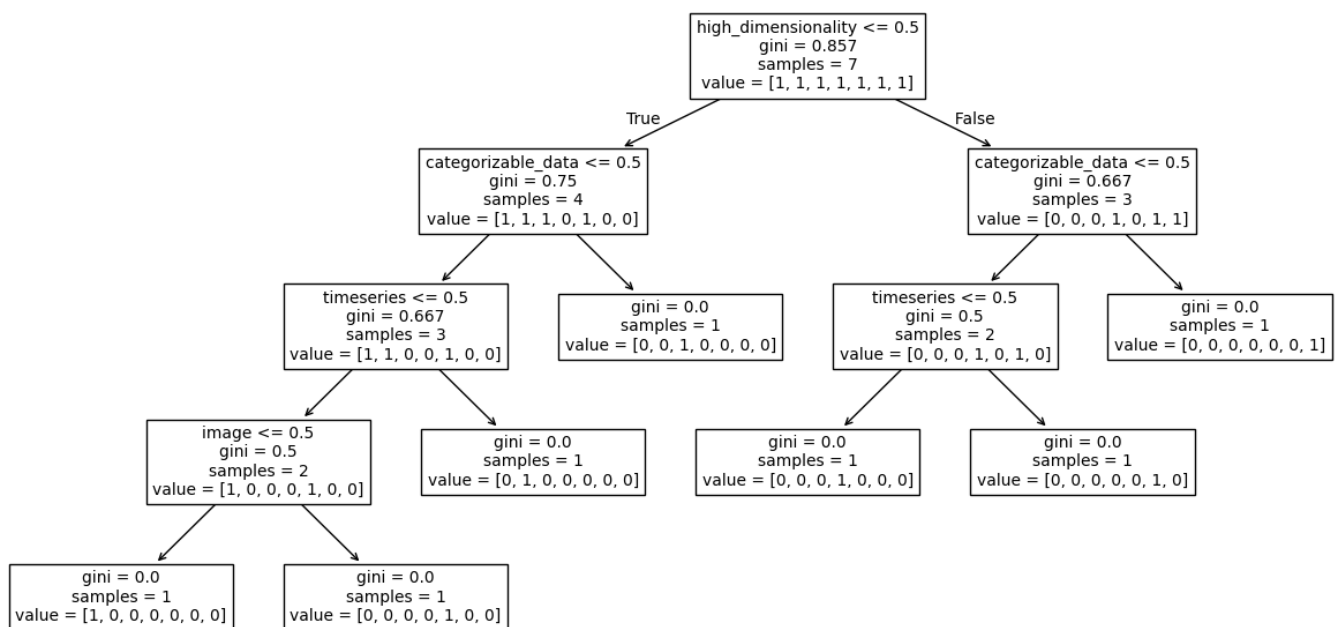


Figure 77: Recommender decision tree classifier visualization

The model contains the following features or input variables, which can take two possible values, 1 or 0:

- *timeseries*: this feature indicates if the dataset is a timeseries or not.



- *categorizable_data*: this feature indicates if the dataset contains data that can be used to categorise it, for example, by gender, zip code, region, academic year, etc.
- *high_dimensionality*: this feature indicates if the dataset has a high number of columns, so that its reduction could be a way of reducing its size.
- *image*: this feature indicates if the dataset contains images.

The model output value or label contains the recommended data sampling method/s. It can take the following possible values:

- 0: *simple_random_sampling* & *systematic_sampling*
- 1: *cluster_sampling* & *stratified_sampling* & *time_series_aggregation*
- 2: *cluster_sampling* & *stratified_sampling*
- 3: *PCA* & *K-means* & *Core-set*
- 4: *GAAL*
- 5: *cluster_sampling* & *stratified_sampling* & *time_series_aggregation* & *PCA* & *K-means* & *Core-set*
- 6: *cluster_sampling* & *stratified_sampling* & *PCA* & *K-means* & *Core-set*
- 7: *simple_random_sampling*
- 8: *systematic_sampling*
- 9: *cluster_sampling*
- 10: *stratified_sampling*
- 11: *time_series_aggregation*
- 12: *PCA*
- 13: *K-means*
- 14: *Core-set*

The model has been trained by providing the input data presented in the following table.

Timeseries	Categorizable data	High dimensionality	Image	Sampling method
0	0	0	0	<ul style="list-style-type: none"> • <i>simple_random_sampling</i> • <i>systematic_sampling</i>



Timeseries	Categorizable data	High dimensionality	Image	Sampling method
1	0	0	0	<ul style="list-style-type: none"> • cluster_sampling • stratified_sampling • Time series aggregation (e.g., average values every 5 minutes)
0	1	0	0	<ul style="list-style-type: none"> • cluster_sampling • stratified_sampling
0	0	1	0	<ul style="list-style-type: none"> • Principal component analysis (PCA) • K-means • Core-set
0	0	0	1	<ul style="list-style-type: none"> • generative adversarial active learning (GAAL)
1	0	1	0	<ul style="list-style-type: none"> • cluster_sampling • stratified_sampling • Time series aggregation (e.g., average values every 5 minutes) • Principal component analysis (PCA) • K-means • Core-set
0	1	1	0	<ul style="list-style-type: none"> • cluster_sampling • stratified_sampling • Principal component analysis (PCA) • K-means • Core-set

Table 3: Input data provided for training the model to recommend sampling techniques



As depicted in the table, depending on the type of the dataset, that is, whether it is a timeseries, or it has categorical data, a high number of columns or if it contains images, the output or label will be a determined series of recommended sampling techniques.

The possible sampling methods that the model recommends are the following:

- Simple random sampling: It consists of selecting the samples randomly, that is, all samples have the same probability of being chosen. The disadvantage of this type of sampling is that there is a chance of picking unrepresentative groupings by chance.²²
- Systematic sampling: This method consists of two steps. In the first step, a simple random selection is carried out, and in the following step, a rule is applied so that every n^{th} item is picked. The disadvantage of this method is that there's a potential risk of introducing bias if there's an unrecognised pattern in the population that aligns with the sampling interval.²²
- Cluster sampling: This method consists of selecting groups or clusters. It can be done by selecting the entire cluster or first selecting a cluster and then picking random samples from that cluster. For example, taking the students belonging to an academic year or the people living in a certain postal code. The disadvantage of this method is that the samples within a cluster might have great similarities, so it could introduce a greater sampling error.²²
- Stratified sampling: This method consists of considering the subgroups in which the data can be divided and then taking a random sample from each group or strata. For example, if we know that 80% of the students are male and 20% are female, and we consider that the gender is an important aspect to consider, we could group the samples by gender and get 80% of the samples from the male group and 20% of the samples from the female group. The disadvantage of this method is that a good knowledge of the dataset stratification is required.²²

²² <https://www.qualtrics.com/experience-management/research/sampling-methods/>



- Time series aggregation: This method could be used when the dataset is a time series. It consists of aggregating the samples considering time intervals, for example, an aggregation of 5 minutes by taking the mean value during that interval.
- Principal Component Analysis (PCA): This method is used to reduce the number of features in a dataset into its essential features, called principal components. Principal components are a few linear combinations of the original variables that maximally explain the variance of all the variables.²³
- K-means: K-Means Clustering is an Unsupervised Machine Learning algorithm which groups an unlabelled dataset into different clusters. It is used to organise data into groups based on their similarity²⁴. K-means clustering can be used in a form of dimensionality reduction where each transformed feature is the distance of a data point from its assigned cluster centre. This technique effectively maps high-dimensional data into a lower-dimensional space based on cluster membership.²⁵
- Core-set: A coreset is a small subset of a large dataset that retains most of the important information in the original dataset. Coresets are often used in machine learning to reduce the time and space required to train and evaluate machine learning models.²⁶
- Generative Adversarial Active Learning (GAAL). This method integrates a generative adversarial network (GAN) with active learning and is used to classify and select images.

27

Random sampling is suitable for uniform distributions, while stratified sampling is used for skewed distributions. If representativeness of subgroups is needed, stratified or cluster sampling is used, and for temporal patterns, stratified, cluster sampling or interval aggregation is preferred. On the

²³ <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>

²⁴ <https://www.geeksforgeeks.org/k-means-clustering-introduction/>

²⁵ <https://medium.com/analytics-vidhya/less-known-applications-of-k-means-clustering-dimensionality-reduction-anomaly-detection-and-908f4bee155f>

²⁶ <https://dataheroes.ai/blog/exploring-different-types-of-reduction-and-sampling-techniques-in-machine-learning/>

²⁷ <https://arxiv.org/pdf/1702.07956>

other hand, if the dataset has a high number of columns, PCA, k-means, or core-set methods could be used to reduce its dimensionality. A dataset containing images could be classified and selected using GAAL active learning. Figure 78 shows a possible data sampling workflow.

Once the recommended sampling techniques are provided, the end user should decide which one of them to use. The user should schedule the sampling procedure with the recommended technique, specifying the execution time, method, and destination path. Since data sampling is not instantaneous, the optimal execution window should be selected, considering key factors such as network traffic, CPU load, and storage availability. This minimises performance bottlenecks by scheduling tasks when resources are least constrained.

The Data Sampling scheduler and data sampling process are not included in this prototype. Companies using DATAMITE will define and develop the best options and algorithms to perform the samplings according to their business requirements. The Data Sampling recommender is not integrated into the DATAMITE frontend since it has been an additional work developed as a possible approach to the dataset size problem.

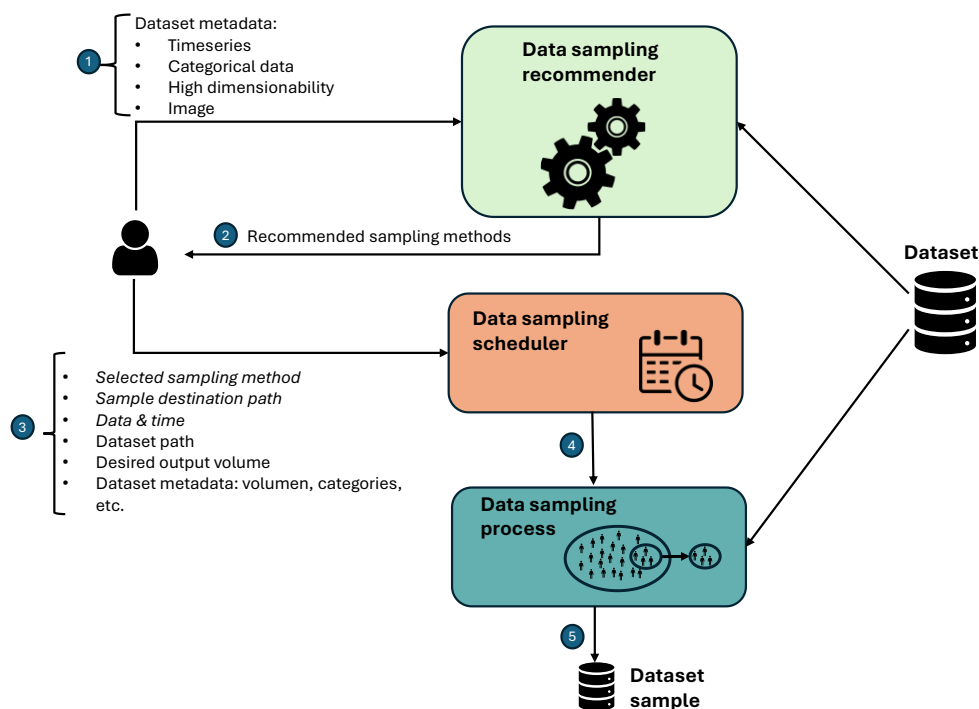


Figure 78: Data sampling workflow



6 Enabling Technologies

This section presents technologies that have been selected to complement the framework internally, assisting in the automation of aspects like pipeline orchestration and secret management. The main rationale behind the integration of these tools is not to reinvent the wheel, but to leverage and integrate existing technologies that can help with the operation of the framework.

At this stage, two enabling technologies have been integrated. First, a pipeline orchestrator, Mage.ai., which is used to simplify the development of pipelines using an open-source tool and avoiding costly licenses. Second, a tool for secret management.

6.1 Pipeline orchestration: Mage.ai

DATAMITE required a tool for creating and orchestrating pipelines, which are used in several processes within the framework, especially in the data ingestion and discovery tasks, or in the profiling. To this purpose, Mage.ai was adopted, assisting in the development, monitoring, and administration of data pipelines and allowing for the smooth integration and transformation of data from many sources. It also provides other interesting features for the framework, such as its capacity to handle errors, process data in real-time, and scale.

As mentioned, in DATAMITE, Mage.ai is utilised to connect to different data sources, retrieve their schemas, and transform them into standard schemas for creating future data products. The use of Mage.ai applies to the following processes:

- Bulk ingestion
- Internal storage in MinIO repositories
- Interaction with Azure storage technologies
- Data discovery in PostgreSQL
- Data discovery in Cassandra
- Data ingestion with MQTT
- Data ingestion with Kafka



- Data ingestion of non-structured data

These processes involve multiple steps, like the interaction with the Data Quality Module to profile data or its registration in the metadata repository. The developments and integration of Mage.ai in DATAMITE can be found in its dedicated [Eclipse repository](#)²⁸.

6.2 Secret management

Within the context of data discovery and data ingestion, DATAMITE must connect to remote databases or publish/subscribe brokers (e.g., MQTT) or other streaming technologies. These connections imply the use of user/pass pairs, which may have to be stored to reuse such connections in the future. For instance, a connection to a database manager can be used to recover a particular set of databases or tables within a database. However, in the future, this same connection could be used to retrieve a different set of data. The same applies to, for instance, a Kafka connection, where the user may be subscribing to certain topics but may want to reuse the connection in the future for a different set of topics.

To store and manage these user/pass pairs, DATAMITE will integrate an open-source secret management tool, such as Hashicorp Vault²⁹.

²⁸ <https://gitlab.eclipse.org/eclipse-research-labs/datamite-project/data-support-tools/mage-tool>

²⁹ <https://www.vaultproject.io/>



7 Conclusions

This document has presented an updated overview of the design and implementation status of the different components within the modules related to WP3. This spans mainly the Data Governance, Quality and Security modules jointly with the Data Governance Tools. The document provides, for each one of these components, the design overview, a summary of the previous status and the description of the final status reached for these developments, as well as links to the Eclipse Repository hosting the development efforts when necessary. It also includes a description of the transversal technologies used by DATAMITE: Mage.ai and Hashicorp. These technologies are leveraged as enablers for different purposes, namely, pipeline orchestration and secret management.

All in all, this document presents a complete summary of the developments that have been undertaken in the components under the umbrella of WP3, which will be included in the final software release of DATAMITE. Finally, note that this document is the final iteration of a series of deliverables. This series started in month 12 and continued in month 18, finishing now and having shown the evolution of the efforts undertaken in WP3.



ANNEX I: APIs summary

This annex presents the most relevant APIs of the modules presented in the document. These APIs are presented in the following tables, including also information regarding the access control, such as the Keycloak client and the permissions. Namely, Table 4 shows the APIs offered by the Data Quality module, from both the QE and the UDRG; Table 5 enumerates the APIs related to the management of pipelines, mostly related to data ingestion and discovery; Table 6 provides a detailed overview of the APIs from the Data Governance module, regarding the creation or manipulation of the different entities in the metadata model; and, finally, Table 7 enumerates the APIs offered by the Data Anonymisation and the Fairness and Data Bias components.

Client	Description			Permission
QE	POST	/evaluate-data	Execute the profiling of a data artifact	WRITE
UDRG	GET	/quality/customMetric	Get all custom metrics	READ
	GET	/quality/customMetric/{dataset_id}	Get custom metrics associated to a dataset	READ
	POST	/quality/customMetric/{dataset_id}	Add a custom metric for a dataset	WRITE
	PUT	/quality/customMetric/{dataset_id}	Update a custom metric for a dataset	WRITE
	DELETE	/quality/customMetric/{dataset_id}	Delete a custom metric of a dataset	DELETE
	GET	/quality/customMetric/{dataset_id}/cm	Get a specific custom metric associated to a dataset	READ
	DELETE	/quality/customMetric/{dataset_id}/all	Delete all custom metrics of a dataset	DELETE
	GET	/quality/customMetric/active/{dataset_id}	Get active status of custom metrics associated to a dataset	READ
	POST	/quality/customMetric/active/{dataset_id}	Update active status of custom metrics associated to a dataset	WRITE
	GET	/quality/customMetric/activeDataset/{dataset_id}	Check if specified dataset has active custom metrics associated	READ
	GET	/quality/customMetric/kpis/required	Get the required KPIs for the custom metrics	READ
	GET	/quality/customMetric/cols/required	Get the required columns for the custom metrics	READ
	POST	/quality/customMetric/interpreter/compile	Evaluate the custom metrics associated to a dataset	READ
	POST	/quality/customMetric/rules/description	Get description of a custom metric	READ
	POST	/quality/rdf_format	Convert custom metrics into RDF format	READ
	GET	/quality/kpilibrary/kpis	Get all kpis	READ

Table 4: Quality module APIs



Client	Description			Permission
Streaming	POST	/streamings/	Create a new streaming	WRITE
	GET	/streamings/	Get all streamings	READ
	GET	/streamings/{streaming_id}	Get streaming by ID	READ
	PUT	/streamings/{streaming_id}	Update streaming by ID	WRITE
	DELETE	/streamings/{streaming_id}	Delete streaming by ID	WRITE
	PUT	/streamings/{streaming_id}/start	Start streaming by ID	WRITE
	PUT	/streamings/{streaming_id}/pause	Pause streaming by ID	WRITE
	PUT	/streamings/{streaming_id}/finish	Finish streaming by ID	WRITE
	PUT	/streamings/{streaming_id}/error	Mark streaming as erroneous by ID	WRITE
	GET	/data_sources/	Get all data sources	READ
Data Sources	POST	/data_sources/	Create a new data source	WRITE
	GET	/streamings/{streaming_id}/artifacts	Retrieve artifacts of a streaming by ID	READ
Artifacts	POST	/streamings/{streaming_id}/artifacts	Create a new artifact	WRITE
	GET	/streamings/{streaming_id}/artifacts/{artifact_id}	Get streaming artifact by ID	READ
	PUT	/streamings/{streaming_id}/artifacts/{artifact_id}/consolidate	Consolidate artifact	WRITE
Refresh artifacts	PUT	/refresh-structured-file/{artifact_id}	Refresh info of a structured file	WRITE
	PUT	/refresh-database/{artifact_id}	Refresh info of a database	WRITE
Enums	GET	/artifact-status	Get Artifact Status Enum	READ
	GET	/streaming-status	Get Streaming Status Enum	READ
	GET	/consolidation-type	Get Consolidation Type Enum	READ
	GET	/file-type	Get File Type Enum	READ
	GET	/source-type	Get Source Type Enum	READ
	GET	/encoding-type	Get Encoding Enum	READ
Health Check	GET	/health	Perform a health check	READ

Table 5: Pipeline-related APIs

Client	Description			Permissions
Glossaries & domains	GET	/terms/{termId}	Retrieves the indicated term type entity	READ
	PUT	/terms/{termId}	Update the term entity	WRITE
	DELETE	/terms/{termId}	Delete the term entity	DELETE
	PUT	/terms/{termId}/entities	Removes term from entities	WRITE
	POST	/terms/{termId}/entities	Assign term to entity	WRITE
	POST	/terms	Create a term entity in the glossary	WRITE
	POST	/terms/{termId}/relations	Create a relation between two terms	WRITE
	DELETE	/terms/relations/{relationId}	deleteTermRelation	DELETE
	GET	/glossaries/{glossaryId}	Retrieves the indicated glossary type entity	READ
	PUT	/glossaries/{glossaryId}	Update the glossary entity	WRITE
	DELETE	/glossaries/{glossaryId}	Delete the glossary entity	DELETE





	GET	/glossaries	Retrieves all the glossaries	READ
	POST	/glossaries	Create a glossary entity in the catalog	WRITE
	GET	/domains/{domainName}	Retrieves the domain	READ
	PUT	/domains/{domainName}	Update the domain	WRITE
	DELETE	/domains/{domainName}	Delete the domain	DELETE
	GET	/domains	Retrieves all the domains	READ
	POST	/domains	Create a domain	WRITE
	POST	/domains/{domainName}/entity/{entityId}	Add the domain to the entity	WRITE
	DELETE	/domains/{domainName}/entity/{entityId}	Remove the domain from the entity	DELETE
dataset	GET	/datasets/{datasetId}	Retrieves the indicated dataset type entity	READ
	PUT	/datasets/{datasetId}	Update the dataset entity	WRITE
	DELETE	/datasets/{datasetId}	Delete the dataset entity	DELETE
	POST	/datasets	Create a dataset entity in the catalog	WRITE
artefacts	GET	/structuredFiles/{structuredFileId}	Retrieves the indicated structured file type entity	READ
	PUT	/structuredFiles/{structuredFileId}	Update the structured file entity	WRITE
	DELETE	/structuredFiles/{structuredFileId}	Delete the structured file entity	DELETE
	POST	/structuredFiles	Create a structured file entity in the catalog	WRITE
	POST	/structuredFiles/{structuredFileId}/qualityMeasurements	Add quality measurements to structured file	WRITE
	GET	/structuredFiles/{structuredFileId}/columns	Retrieves columns from the file and their quality	READ
	GET	/databases/{databaseId}	Retrieves the indicated database type entity	READ
	PUT	/databases/{databaseId}	Update the database entity	WRITE
	DELETE	/databases/{databaseId}	Delete the database entity	DELETE
	POST	/databases	Create a database entity in the catalog	WRITE
	GET	/databases/{databaseId}/tables	Retrieves table names with columns and quality	READ
	POST	/databases/{databaseId}/tables	Add table entities to the database entity	WRITE
	GET	/databases/uniqueAttribute	Retrieves the database by the indicated unique attribute	READ
	GET	/tables/{tableId}	Retrieves the indicated table type entity	READ
	PUT	/tables/{tableId}	Update the table entity	WRITE
	POST	/tables/{tableId}	Add quality measurements to database table	WRITE
	DELETE	/tables/{tableId}	Delete the table entity	DELETE
	GET	/tables/{tableId}/columns	Retrieves columns from the table and their quality	READ
	GET	/columns/{columnId}	Retrieves the indicated column type entity	READ
	PUT	/columns/{columnId}	Update the column entity	WRITE
	DELETE	/columns/{columnId}	Delete the column entity	DELETE
	POST	/columns	Create column entity for the dataset	WRITE
	GET	/unstructuredFiles/{unstructuredFileId}	Retrieves the indicated unstructured file type entity	READ
	PUT	/unstructuredFiles/{unstructuredFileId}	Update the unstructured file entity	WRITE
	DELETE	/unstructuredFiles/{unstructuredFileId}	Delete the unstructured file entity	DELETE



	POST	/unstructuredFiles	Create a unstructured file entity in the catalog	WRITE
	POST	/unstructuredFiles/{unstructuredFileId}/qualityMeasurements	Add quality measurements to unstructured file	WRITE
	GET	/videos/{videoDistributionId}	Retrieves the indicated video distribution type entity	READ
	PUT	/videos/{videoDistributionId}	Update the video distribution entity	WRITE
	DELETE	/videos/{videoDistributionId}	Delete the video distribution entity	DELETE
	POST	/videos	Create a video distribution entity in the catalog	WRITE
	POST	/videos/{videoDistributionId}/qualityMeasurements	Add quality measurements to video distribution	WRITE
	GET	/images/{imageDistributionId}	Retrieves the indicated image distribution type entity	READ
	PUT	/images/{imageDistributionId}	Update the image distribution entity	WRITE
	DELETE	/images/{imageDistributionId}	Delete the image distribution entity	DELETE
	POST	/images	Create an image distribution entity in the catalog	WRITE
	POST	/images/{imageDistributionId}/qualityMeasurements	Add quality measurements to image distribution	WRITE
	GET	/audios/{audioDistributionId}	Retrieves the indicated audio distribution type entity	READ
	PUT	/audios/{audioDistributionId}	Update the audio distribution entity	WRITE
	DELETE	/audios/{audioDistributionId}	Delete the audio distribution entity	DELETE
	POST	/audios	Create an audio distribution entity in the catalog	WRITE
	POST	/audios/{audioDistributionId}/qualityMeasurements	Add quality measurements to audio distribution	WRITE
	POST	/processes/dbSnapshot	Create a dbSnapshot process	WRITE
entity	GET	/lineage/{entityId}	Get lineage from an entity	READ
	POST	/quality/{entityId}/score	Set quality score	WRITE
data product	GET	/dataProducts/{dataProductId}	Retrieves the indicated data product type entity	READ
	PUT	/dataProducts/{dataProductId}	Update the data product entity	WRITE
	DELETE	/dataProducts/{dataProductId}	Delete the data product entity	DELETE
	POST	/dataProducts	Create a data product entity	WRITE
	GET	/dataProducts/{dataProductId}/fairness	Get fairness metrics from a data product entity	READ
	POST	/dataProducts/{dataProductId}/fairness	Add fairness metrics to a data product entity	WRITE
	POST	/dataProducts/{dataProductId}/aggregations	Add aggregations to a data product entity	WRITE
Data publications	GET	/publications/{publicationId}	Retrieves the indicated publication type entity	READ
	PUT	/publications/{publicationId}	Update the publication entity	WRITE
	DELETE	/publications/{publicationId}	Delete the publication entity	DELETE
	GET	/publications	Retrieves all the publications	READ
	POST	/publications	Create a publication for the data product	WRITE
Search	POST	/search	Search entities	READ



Storage	GET	/search/terms/{text}	Get terms by text	READ
	GET	/metrics/count	Returns the number of entities in the catalog. You can specify the types to be counted.	READ
	GET	/artifacts/{artifactId}/files/{fileName}/file-path	Get the file path of the artifact in the storage specific format	READ
	GET	/artifacts/{artifactId}/files/{fileName}/file-url	Get the file url of the artifact in the storage specific format	READ
	GET	/artifacts/{artifactId}/files/{fileName}/location	Get the download location of the artifact in the storage specific format	READ
	GET	/artifacts/{artifactId}/files/{fileName}/upload-location	Get the upload location of the artifact in the storage specific format	WRITE
	POST	/artifacts/{artifactId}/files/{fileName}/multipart	Create a multipart upload of the artifact in the storage specific format	WRITE
	GET	/artifacts/{artifactId}/files/{fileName}/upload/{uploadId}	List the uploaded parts of a multipart artifact upload	WRITE
	GET	/artifacts/{artifactId}/files/{fileName}/upload/{uploadId}/part/{partNumber}	Get the upload location of the artifact part	WRITE
	POST	/artifacts/{artifactId}/files/{fileName}/upload/{uploadId}/complete	Complete the multipart upload of the artifact	WRITE
	DELETE	/artifacts/{artifactId}/files/{fileName}/upload/{uploadId}	Abort the multipart upload of the artifact	WRITE
	DELETE	/artifacts/{artifactId}/files/{fileName}	Delete the artifact	DELETE

Table 6: Data Governance module APIs

Client	Description			Permissions
Fairness	GET	/project/get_projects	Get the fairness analysis projects	READ
	GET	/project/get_project/{project_id}	Get the fairness analysis project information	READ
	POST	/project/add_project	Creates a new fairness project.	WRITE
	POST	/project/add_project_with_id	Creates a new fairness project. The accepted payload has name, description and data product id	WRITE
	DELETE	/project/delete_project/{project_id}	Deletes an existing project	DELETE
	PUT	/project/update_project/{project_id}	Updates an existing project. The accepted payload has name and description	WRITE
Anonymization	POST	/anonymization/load_data	Upload a dataset to DATAMITE framework from pc	WRITE
	GET	/anonymization/get_data_product	Upload saved data product to DATAMITE framework from database	READ
	GET	/anonymization/get_saved_data_from_anonymization	Get all data of all saved data products	READ
	GET	/anonymization/get_all_metadata	Get all metadata of all saved data products	READ
	GET	/anonymization/get_all_custom_lists	Get all custom lists	READ
	POST	/anonymization/add_custom_list_sensitive_columns	Add a custom list to the database	WRITE
	PUT	/anonymization/update_custom_list_sensitive_columns	Update a custom list	WRITE



DELETE	/anonymization/delete_custom_lists	Delete a custom list	DELETE
POST	/anonymization/identify_columns	Identifies sensitive columns (from custom lists or predefined-general)	READ
POST	/anonymization/post_validate_columns	Validate selected sensitive column/s for privacy models	READ
POST	/anonymization/post_column_info	Sensitive column selection	READ
POST	/anonymization/post_anonymization_info	Select anonymization technique, deanonymization and privacy model selection for every sensitive column and anonymize them	READ
POST	/anonymization/save_anonymized_data_into_anonymization_table	Save anonymized data	WRITE
POST	/anonymization/update_data_product_with_anonymized_data	Update data product with anonymized data	WRITE
DELETE	/anonymization/delete_anonymized_datasets	Delete anonymized datasets from the database	DELETE
GET	/anonymization/download/{dataset_id}	Download anonymized dataset in various formats (XLSX, CSV, ZIP, JSON)	READ
GET	/anonymization/get_deanonymization_columns	Available columns that can be deanonymize from saved dataset	READ
POST	/anonymization/post_column_to_deanonymize	Deanonymize columns	WRITE

Table 7: Data Support Tools - APIs for Fairness and Anonymisation tools.

