SEcure Decentralised Intelligent Data MARKetplace

D5.3 Integrated releases of the SEDIMARK platform. Second version

Document Identification	
Contractual delivery date:	31/12/2024
Actual delivery date:	15/01/2025
Responsible beneficiary:	WINGS
Contributing beneficiaries:	WINGS, UC, EVIDEN, LINKS, SURREY, EGM, SIE, NUID UCD, INRIA
Dissemination level:	PU
Version:	v1.0
Status:	Final

Keywords:

Three parallel streams, intelligence, orchestration, marketplace, decentralisation, AI assets, software components, functional/non-functional requirements, continuous integration, continuous delivery, SEDIMARK platform, backend and frontend integration

This document is issued within the frame and for the purpose of the SEDIMARK project. This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No. 101070074. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

Document Information

Document Identification			
Related WP	WP5	**Related Deliverables(s):**	SEDIMARK_D5.1, SEDIMARK_D5.2
Document reference:	SEDIMARK_D5.3	**Total number of pages:**	79

List of Contributors	
Name	**Partner**
Panagiotis Vlacheas Grigorios Koutantos	WINGS
Pablo Sotres Luis Sánchez Juan Ramón Santana Jorge Lanza	UC
Stefan Jarcau Gabriel Danciu Septimiu Nechifor	SIE
Maxime Costalonga Cesar Caramazana Zarzosa Joaquin Garcia	ATOS/EVIDEN
Elias Tragos Erika Duriakova Diarmuid O'Reilly-Morgan Honghui Du	NUID UCD
Tarek Elsaleh	SURREY
Shahin Abdoul Soukour Nikolaos Georgantas	INRIA
Franck Le Gall Thomas Bousselin	EGM

List of Contributors	
Name	**Partner**
Michele Festa Alberto Carelli	LINKS

Document History			
ToCs	13/11/2024	WINGS	First draft of ToCs
0.1	15/11/2024	UC	Added content to Sections 2 and 3
0.11	26/11/2024	WINGS	Initial version with reconstructed ToCs based on UC's suggestions
0.2	2/12/2024	SIE	Added content to Section 3 MVI
0.3	3/12/2024	SIE	Updated content to Section 3 MVI
0.31	3/12/2024	WINGS	Introduction and Executive Summary
0.32	4/12/2024	SIE	Update Section 3
0.4	5/12/2024	WINGS	Update Section 4
0.41	6/12/2024	UCD	Update Section 3
0.5	10/12/2024	SIE, ATOS/EVIDEN	Structural and content changes in Section 3
0.51	11/12/2024	ATOS/EVIDEN	Added content to Section 3.
0.6	11/12/2024	SURREY, SIE, ATOS/EVIDEN	Update content to Section 3.3, Section 4
0.6	13/12/2024	INRIA	Update Section 3.2.1.3
0.61	12/12/2024	WINGS	Fill Section 6 and acronyms table
0.62	13/12/2024	UC	Fill Section 2
0.63	16/12/2024	UC, LINKS	Updated content to Sections 3 and 4
0.64	17/12/2024	WINGS	Update content to Section 3
0.7	17/12/2024	EGM	Update content to Section 3 (NGSI-LD Broker integration, Connection to MinIO)
0.71	18/12/2024	WINGS	Update Section 5

0.72	19/12/2024	SIE, WINGS, UC	Integration specification screenshots from GitHub
0.75	20/12/2024	WINGS	Formatting, template fitting, cleaning, etc.
0.8	14/01/2024	ATOS/EVIDEN	Quality Review From
1.0	15/01/2025	ATOS/EVIDEN	FINAL VERSION TO BE SUBMITTED

Quality Control		
Role	**Who (Partner short name)**	**Approval date**
Reviewer 1	John Tsogias & Nikos Babis (MYT)	07/01/2025
Reviewer2	Thomas Bousselin (EGM)	06/01/2025
Quality manager	María Guadalupe Rodríguez (EVIDEN)	14/01/2025
Project Coordinator	Miguel Angel Esbri (EVIDEN)	15/01/2025

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / Acronym	Description
AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Service
CI/CD	Continuous Integration and Continuous Delivery/Continuous Deployment
CSV	Comma Separated Values
CWL	Common Workflow Language
DID	Decentralised Identifier
DLT	Distributed Ledger Technology
DSP	Data Space Protocol
Dx,y	Deliverable number y belonging to WP x
EDC	Eclipse Dataspace Components
EVM	Ethereum Virtual Machine
GUI	Graphical User Interface
H-REQ	High-priority Requirements
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identity Document
IOTA	Internet of Things Application
JSON	JavaScript Object Notation
ML	Machine Learning
MVI	Minimum Viable Intelligence
MVM	Minimum Viable Marketplace
NFT	Non-fungible Token
NGSI-LD	Next Generation Service Interfaces for Linked Data
ODRL	Open Digital Rights Language

Abbreviation / Acronym	Description
P2P	Peer to Peer
PoC	Proof of Concept
REST	Representational State Transfer
SC	Smart Contract
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
VC	Verifiable Credential
VDR	Verifiable Data Registry
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XLS	Microsoft Excel Spreadsheet
YAML	Yet Another Markup Language

Executive Summary

The current document is the third deliverable of WP5 and reports the results of Task 5.2 activities regarding continuous platform integration. Building upon the foundational work presented in SEDIMARK_D5.2 [1], this deliverable focuses on the enhancement and refinement of the platform's capabilities through the incorporation of Minimum Viable Marketplace (MVM) and Minimum Viable Intelligence (MVI) functionalities. It emphasises the integration of decentralised data and service-sharing frameworks, advanced AI-driven tools, secure components based on Distributed Ledger Technology (DLT), APIs, and platform-wide testing. These components are designed to address medium and high-priority requirements for interoperability, trustworthiness, data quality, etc, and they were developed and described in WP3 "Distributed data quality management and interoperability" and WP4 "Secure data sharing in a decentralised Marketplace".

The three streams will be analysed by the leaders and their contributors on the following topics:

- Sub-stream breakdown and component specification: Scope of the stream, task by task description, updates in the components, meaningful visualisation and solid text accompanying the figures.
- Integration specifications: Integration steps (inter/intra component communication, setup monitoring and logging, CI/CD implementation) accompanied by GitHub screenshots.

The second release achieves the following milestones:

- Successful integration of core enablers such as AI Orchestrator, Data Space Enabler and DLT enabler.
- Enhanced implementations focusing on participant onboarding, data quality, offering lifecycle management, asset exchange, and distributed AI training.
- Streamlined APIs and interfaces, ensuring seamless communication across platform components and external participants.
- Deployment-ready solutions validated against real-world use cases to ensure scalability and reliability.

Following the structure of SEDIMARK_D5.2, it is important to ensure that the requirements specified in WP2 architecture and Tasks T2.1-T2.4, summarised with the SEDIMARK_D2.3 [2], are fulfilled at all implementation phases. To serve this need, there are updated tables correlating all the medium-priority recommended requirements (M-REC) that were promised to be fulfilled for the second version of the platform, added to the existing high-priority requirements (H-REQ). The target is to monitor the status of fulfilment and in which stream they are addressed. The table covers both the functional and non-functional requirements.

The final part of the document provides an overview of the final integrated release of the SEDIMARK platform. The idea is to provide all the toolbox functionalities, where all components are in place and the system is optimised for performance purposes. Also, no hard coding is needed, and all the kinds of requirements will be fulfilled. The integration in this way will consider the timeplan for releasing the SEDIMARK integrated platform, as described in SEDIMARK_D5.1 [3]. Through continuous iterations, SEDIMARK is positioned as a robust, secure, and efficient decentralised marketplace for data and services.

1 Introduction

1.1 Purpose of the document

The main purpose of the deliverable is to report the outcomes of the second integrated release of the SEDIMARK platform. This version builds on the architectural foundations established in SEDIMARK_D5.1 and SEDIMARK_D5.2 and defined in SEDIMARK_D2.3, incorporating refined functionalities and addressing gaps identified during the first release. Leveraging the seven independent PoCs scenarios developed in the first version, now the consortium introduces three complementary streams; the Minimum Viable Marketplace (MVM), the Minimum Viable Intelligence (MVI) and the combined Minimum Viable Marketplace - Minimum Viable Intelligence (MVM-MVI) stream which connects the backend and the frontend view. All streams together, cover the toolbox functionalities required for the second version of the platform. The overall goal is to present a version of the platform that encompasses all fundamental functionalities and addresses both medium and high-priority requirements. It serves as a comprehensive reference for understanding the progress in platform integration and highlights challenges and lessons learned.

1.2 Relation to other work packages and tasks

This deliverable is the outcome of Task 5.2 (Platform continuous integration) and is the continuation of the work done during the second year of the project especially in Task T5.1 (Integration and Evaluation plan and methodologies). SEDIMARK_D5.3 is a follow-up deliverable of SEDIMARK_D5.2, elaborating on the scenarios developed there but now the work is being done in three parallel streams. The work presented in the document is strongly related to the components and tools developed in WP3 and WP4, which involve the technical aspects of the platform's development and create the overall decentralised marketplace, based on the architecture design of Task T2.3, the interfaces specified in Task T2.4 and final version of the architecture defined in SEDIMARK_D2.3. The output of SEDIMARK_D5.3 will also be used as input to the upcoming activities of the remaining tasks (Task T5.3, Task T5.4) of WP5 for the three integrated releases of the SEDIMARK platform which will be presented in three phases (M18-Mar. 2024, M27-Dec. 2024, M36-Sep. 2025) and analysed in the current deliverable SEDIMARK_D5.3 (Integrated releases of the SEDIMARK platform. Second version), and forthcoming deliverables SEDIMARK_D5.4 (Integrated releases of the SEDIMARK platform. Final version). The final deliverable of WP5 will document further progress and refinements to the platform. This gradual platform deployment allows beneficiaries to gain valuable insights into performance and make any necessary adjustments or improvements.

1.3 Structure of the document

This document is structured in 6 major chapters:

Chapter 1 is the current chapter, providing context, purpose, and connections to other tasks.

Chapter 2 details the new integration approach, status and components.

Chapter 3 is the main chapter of the deliverable and focuses on the conversion of the independent PoCs scenarios to the three different streams that will be implemented and their integration status.

Chapter 4 checks the correlation of the streams with the medium and high-priority requirements of the revised architecture.

Chapter 5 outlines the roadmap for the final integration release of the SEDIMARK platform.

Chapter 6 concludes the document, summarising the achievements and the next steps in alignment with the objectives and the goals of the work package.

2 Second integrated release of SEDIMARK platform

The second release of the SEDIMARK platform is a significant milestone in our project. During the previous period, the platform's capabilities were showcased through the seven independent scenarios, each serving as a proof-of-concept for the implementation and results of the different concepts explored within the project, that were identified and developed for the first release. The focus of the work for the second release has been on the integration of such capabilities so that this is the first step towards showcasing an actual implementation of the system acting as a single artifact. This integrated system aims to demonstrate the versatility and robustness of our architecture enabling support for more complex scenarios that involve several steps of the lifecycle of an asset within the SEDIMARK Marketplace.

As the SEDIMARK platform evolves and improves throughout successive iterations, its components will be incrementally enhanced and refined to improve performance and functionality. The main goal of the work carried out for this second release has been to ensure that previously independent functionalities can now run sequentially from the same set of interfaces. This approach creates a more streamlined and cohesive workflow. In this regard, the components that were behind those scenarios have been enhanced to improve their performance and extend their functionalities, but, mainly, they have been tuned to be integrated with the rest of components (which were supporting another of the initial PoCs scenarios).

Last but not least, another key aspect that has been incorporated into the second release of the SEDIMARK platform has been the mechanisms to easily deploy it. In this sense, considering the system architecture that has been defined in SEDIMARK_D2.3 the deployment of the SEDIMARK platform has two main aspects. On the one hand, the Baseline Infrastructure, which is, essentially, the nodes supporting the Distributed Ledger Technology network. In this regard, a network of nodes supporting Layer 1 and Layer 2 of the IOTA Tangle has been deployed at the premises of several partners. This will be the network supporting the execution of the pilot cases within the SEDIMARK project, but easy-to-follow documentation has been created in case a different Baseline Infrastructure needs to be created. On the other hand, the SEDIMARK Toolbox is the artifact that integrates most of the developments that have been carried out in WP3 and WP4. Any participant willing to interact within the SEDIMARK Marketplace is required to deploy their own instance of the SEDIMARK Toolbox. Thus, the ease of the SEDIMARK Toolbox deployment has been, precisely, one of the key objectives and focuses for the second release of the SEDIMARK platform.

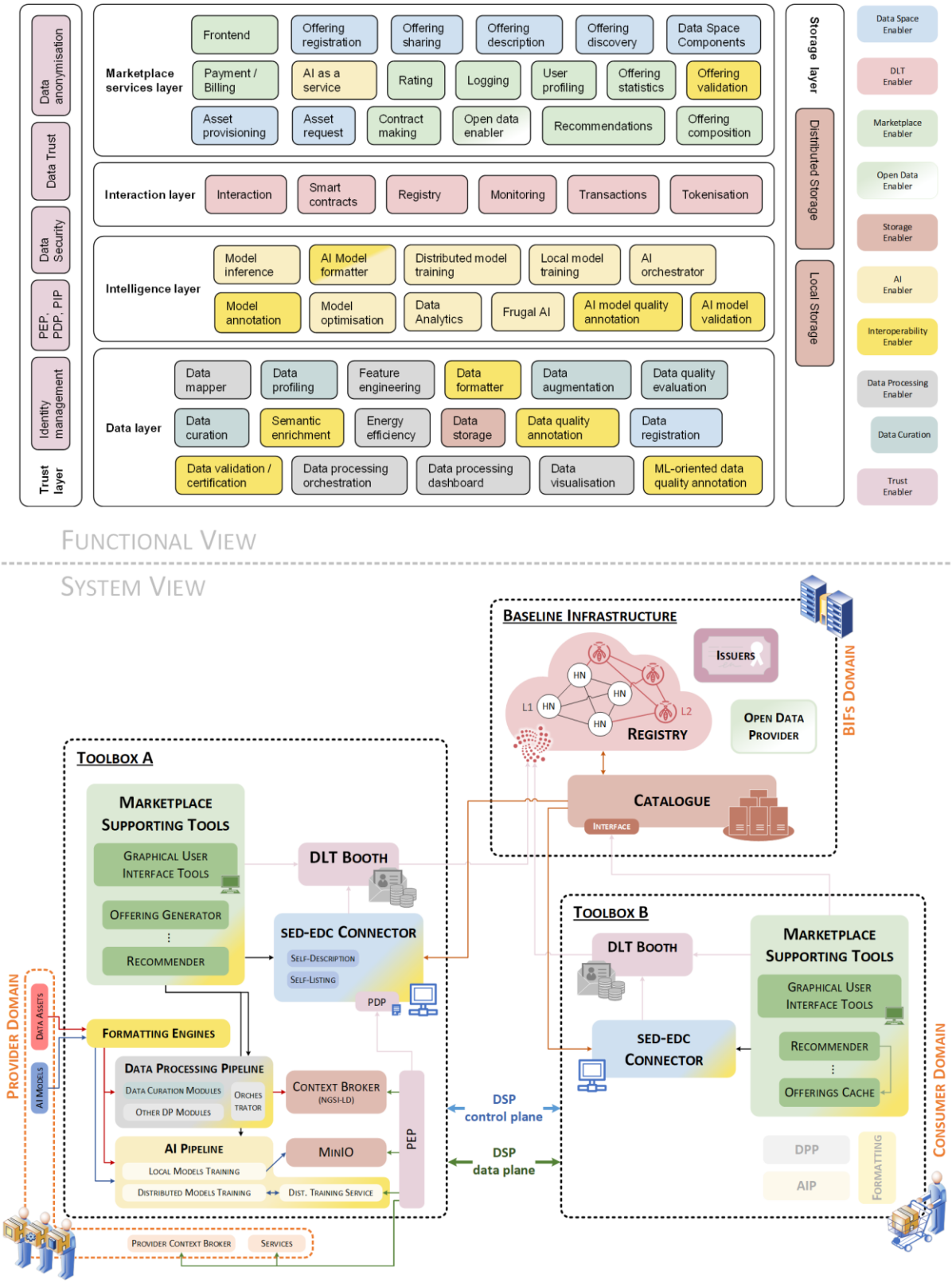


Figure 1 General view of the functional components and the current system view

3 From seven independent scenarios PoCs to three parallel streams

In the previous phases of the SEDIMARK platform development, seven independent scenarios PoCs were introduced instead of consolidating into an integrated platform right away, focusing on validating individual functionalities of the platform and components of the architecture. These PoCs demonstrated the feasibility of critical platform features addressing high-priority requirements. To implement each scenario, a subset of the components under development in the project were integrated, thereby presenting an initial version of the platform. For your reminder, the PoCs scenarios were listed as the following shown below in the bullets and Figure 2 but specific details can be found in the deliverables SEDIMARK_D5.1 and SEDIMARK_D5.2.

- Data quality improvement
- Offering lifecycle
- Participants onboarding
- Asset (Data) exchange
- AI-related scenarios
- GUIs
- Open data enabler

However, as the platform matured, it became evident that an integration approach was required to achieve a more solid methodology and exploit the full potential of the architecture. To serve this purpose, the consortium proposed the transition from independent PoCs to three parallel streams; Minimum Viable Marketplace (MVM), Minimum Viable Intelligence (MVI), and their combined functionality Minimum Viable Marketplace - Minimum Viable Intelligence (MVM-MVI) which correlates the backend and the frontend integration. This integration ensures that platform components can operate cohesively, in an interoperable, scalable and trustworthy manner, addressing complex use cases while meeting medium and high-priority requirements promised for the second version of the platform.

- **MVM**: Focuses on providing the foundational marketplace functionalities, including participant onboarding, offering registration, and secure data exchange.
- **MVI**: Implements AI-driven capabilities for data preparation and formatting, data analytics, local and distributed model training and optimisation, asset exchange and others, emphasizing decentralisation and privacy.
- **Backend and frontend integration (MVM-MVI)**: Combines marketplace and intelligence functionalities, creating a unified framework where participants can not only share and discover data but also leverage AI tools for insights and optimisation in a user-friendly manner.

This structured approach transforms the SEDIMARK platform from a collection of standalone solutions into an integrated ecosystem capable of supporting diverse pilot applications. The next subsections outline the process and the sub-streams of each stream and how they operate and interact. A direct link to the GitHub repository is placed when required for the reader's ease.

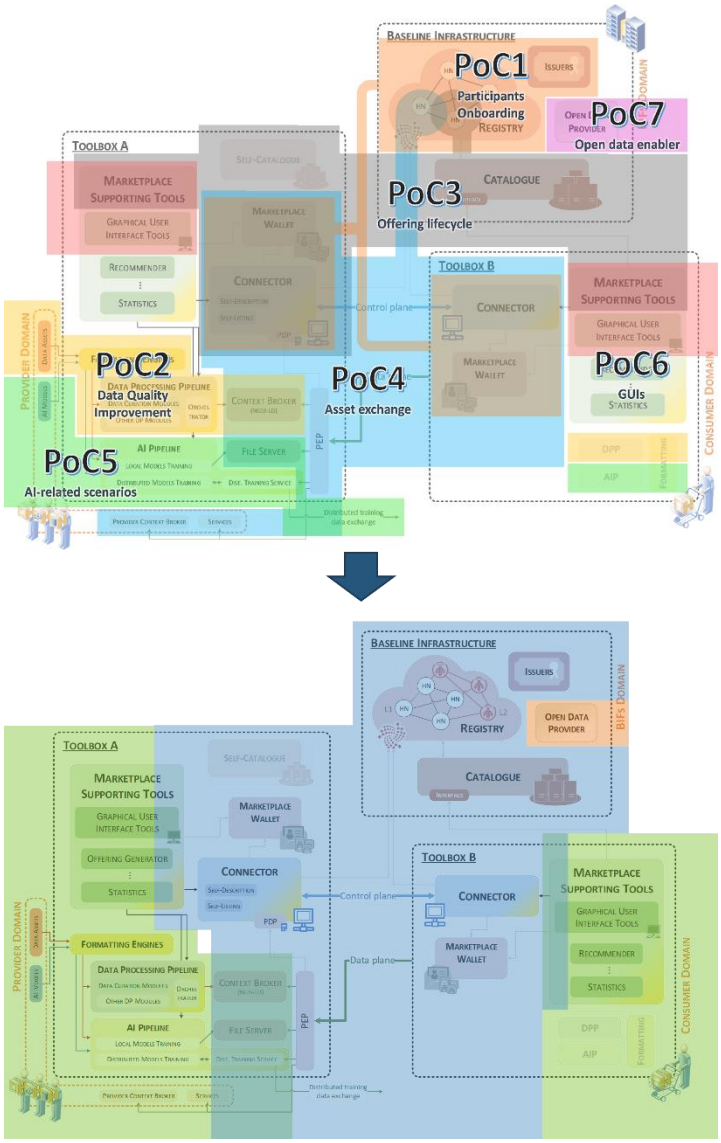


Figure 2 Transferring the seven Proof-of-Concept scenarios to three parallel streams

3.1 Minimum Viable Marketplace (MVM)

The Minimum Viable Marketplace (MVM) encompasses the basic set of components required to facilitate the offering lifecycle and secure P2P asset exchange within the SEDIMARK ecosystem. This framework ensures that all necessary elements are in place to support the basic operations and interactions between participants, enabling a functional and efficient marketplace environment. The marketplace stage includes critical processes such as offering registration and negotiation, allowing participants to establish agreements to regulate transaction terms. The subsequent asset exchange mechanisms are supported by secure protocols that ensure the confidentiality and integrity of all communications. Additionally, the MVM incorporates participant onboarding processes, where providers and consumers generate credentials and cryptographic keys to ensure the security and authenticity of their interactions.

As a result, by focusing on the bare minimum requirements, the MVM ensures a secure and effective marketplace, enabling smooth and trustworthy interactions between participants and providing a solid foundation for future growth and development. Moreover, the emphasis on security and trustworthiness helps build a robust and resilient marketplace that can adapt to evolving needs and challenges.

3.1.1 Sub-streams breakdown and components specification

The integration work carried out to achieve MVM can be divided into three distinct sub-streams: participant onboarding, offering management, and asset exchange.

3.1.1.1 Participant onboarding

Participant onboarding includes the various steps that each new user must follow to become a SEDIMARK participant and to generate and obtain the necessary cryptographic objects/credentials required to establish trusted and secure interactions in the marketplace context. Figure 3 illustrates the various components involved in this process. From the user's perspective, the SEDIMARK Marketplace Frontend (depicted in the lower part of the figure) provides a simplified interface that facilitates interaction with the underlying complex system.

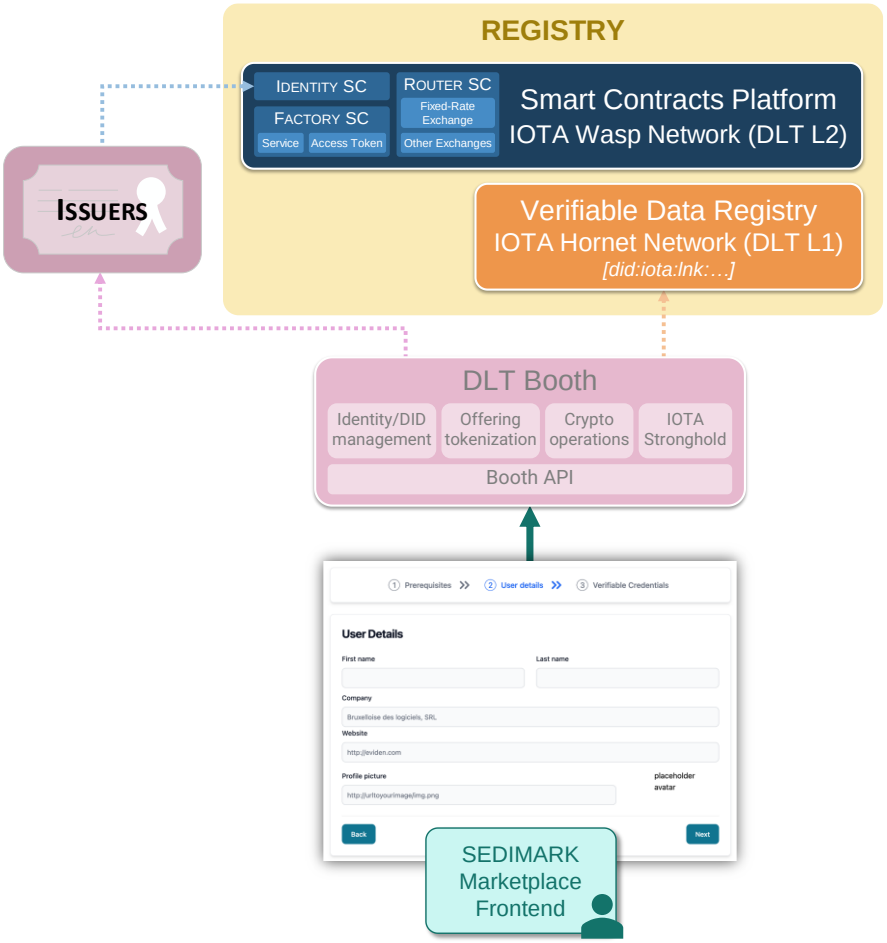


Figure 3 Participant onboarding components

Once the user has supplied their profile information and completed the required steps, the SEDIMARK frontend will directly interact with the DLT Booth component, leveraging the

functionalities offered by the Booth API. The DLT Booth is designed to streamline the management of cryptographic operations, while also providing a secure storage for credentials. In this regard, the available operations include the generation of the necessary identity key pairs, the DID document generation and management, together with the handling of its storage in the Verifiable Data Registry (VDR), and the interaction with the Issuer to obtain a valid SEDIMARK Verifiable Credential (VC). This credential together with the participant's private keys are subsequently stored within its IOTA stronghold vault. Furthermore, issuers also engage with the Smart Contracts Platform, specifically the Identity Smart Contract (SC), to enable credential revocation capabilities. The role of this SC is depicted in Figure 4, where the whole SEDIMARK smart contract architecture [4] is represented.

Both the SEDIMARK Marketplace Frontend [5] and the DLT Booth [6] components are part of the SEDIMARK toolbox, which is deployed on the participant domain per their particular relevant security policies and restrictions. The Registry [7] and Issuer [8], however, are components deployed by the Baseline Infrastructure Facilitators and therefore reside in the cloud. For this reason, communication between the first is considered to happen in a local context.

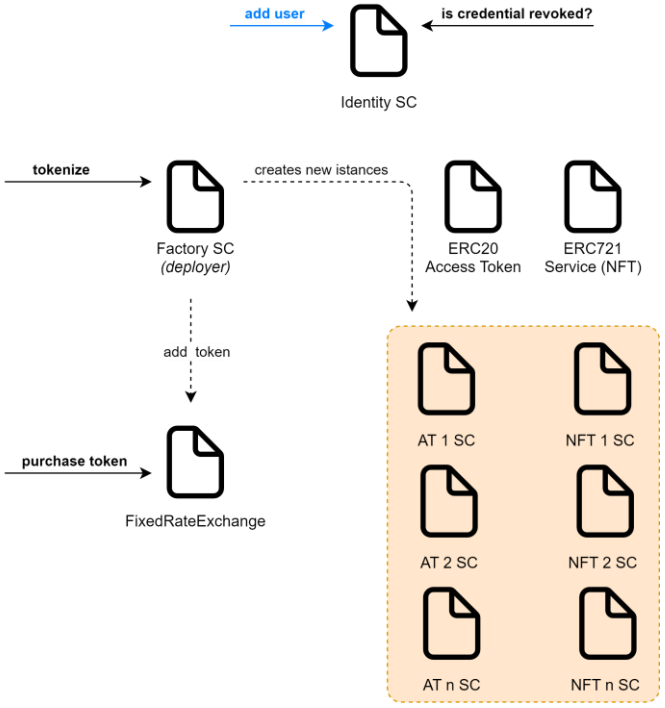


Figure 4 SEDIMARK smart contract architecture

3.1.1.2 Offering management

The offering management sub-stream focuses on controlling the life-cycle of all offerings within the system, ensuring their trustworthy registration and tokenisation and enabling their accessibility by any potential consumer in the Marketplace. Figure 5 illustrates the various components associated with this functionality, organised by the specific stage they are involved in i) creation, modification and withdrawal; ii) registration; iii) population; and iv) discovery.

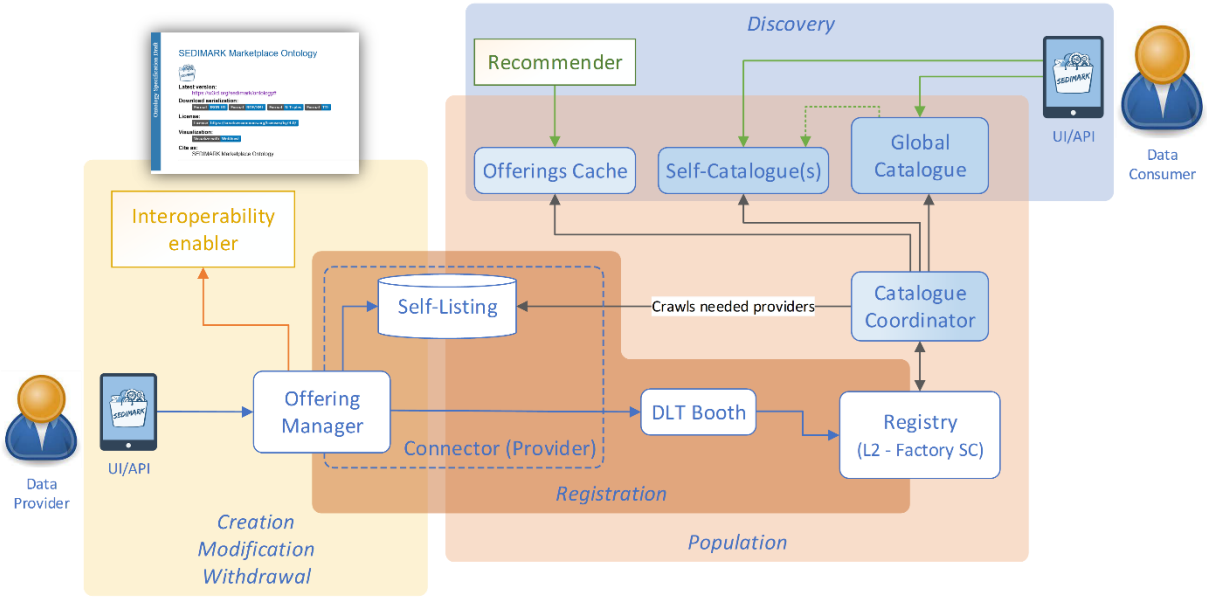


Figure 5 Offering management components

Once the offering document has been created as a result of the interactions in the upper layers, the Offering Manager [9] component acts as a local gateway to initiate offering registration. Initially, it leverages the interoperability enabler to perform a syntactic and semantic validation of the provided offering. After successful validation, the document is hashed and stored in the Self-Listing for later retrieval. The result of such hash, together with the public URL of the stored offering, serves as input to the Booth API operation responsible for tokenizing the offering using the Factory SC from the Registry. The result of this process is the generation of an ERC721 token (NFT) representing the offering as a whole together with a set of ERC20 tokens representing the ownership of a contract for that particular offering, as depicted in Figure 4.

As part of the offering tokenisation mechanism, a series of events are generated by the Smart Contract Platform. A Catalogue Coordinator can consume these events to synchronize the content of the Registry with any Offering Catalogue. In particular, SEDIMARK provides a reference implementation of such a component [10], which is detailed in Figure 6. In addition to real-time event-based synchronisation, any Catalogue Coordinator can query the Factory SC to obtain a list of all existing NFTs that represent the existing offerings in the Marketplace. With this information, the coordinator can later crawl each provider's self-listing to retrieve the complete offering description and validate the hash to ensure it has not been modified since it was registered. How each specific Catalogue Coordinator then populates their catalogue is technology-dependent. The SEDIMARK Catalogue [11] relies on a central global triple store to support SPARQL-based semantic queries, so the Catalogue Coordinator sends every newly discovered offering to that endpoint. This same endpoint is targeted by the SEDIMARK Marketplace Frontend for supporting the discovery stage. It is interesting to note that another

Catalogue Coordinator is envisioned to generate a local Offerings Cache exploited by the Recommender during the discovery stage.

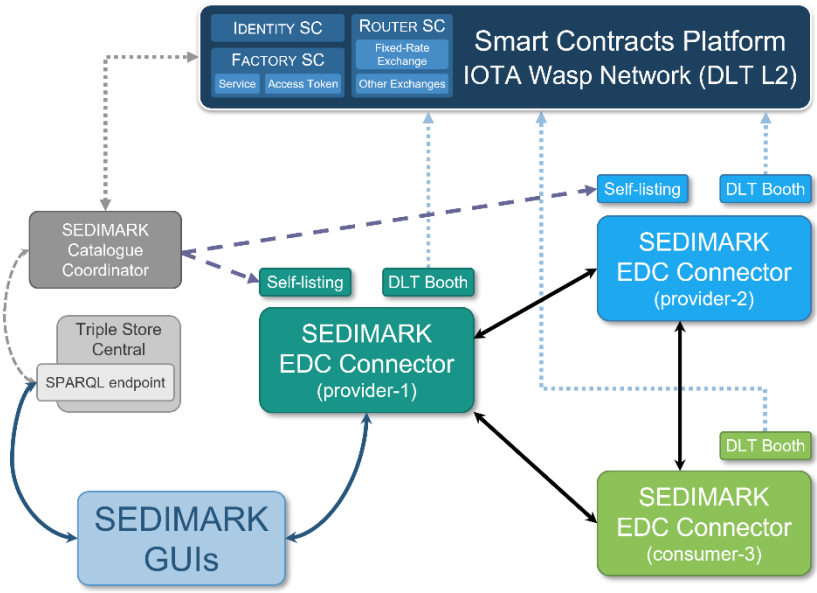


Figure 6 Detail of components involved in the offering population and discovery stages

3.1.1.3 Asset exchange

The asset exchange sub-stream encompasses all the mechanisms that facilitate and guarantee the secure exchange of assets between pairs of SEDIMARK participants. In this sense, and by leveraging cryptographic techniques and DLT technologies, this stage ensures that exchanges are recorded immutably, providing a verifiable trail of transactions in a distributed ledger that enhances the trustworthiness of the system.

Figure 7 shows the set of components and interactions that collectively enable the provision of asset exchange-related functionalities. This diagram provides a comprehensive overview of the subsystem, highlighting the different elements involved, which are deployed both as part of the SEDIMARK baseline infrastructure as well as on every individual participant domain.

The main elements of this sub-stream are as follows: the SEDIMARK dataspace connector, the DLT Booth, the SEDIMARK smart contract architecture, and the various backends, also known as data sinks, which are used by providers to store the actual assets.

The SEDIMARK Dataspace Connector enables dataspace-related functionalities within the MVM, serving as a gateway through which participants can interact among them within the dataspace ecosystem. This way, by leveraging standardised interfaces, the SEDIMARK Dataspace Connector ensures interoperable communication and data exchange between different entities.

This component is based on the well-known Eclipse Dataspace Components (EDC) framework and its connector, taking advantage of its robust and reliable control plane. The control plane is compatible with the Data Space Protocol (DSP), ensuring that all interactions within the dataspace adhere to established standards and protocols. Additionally, the EDC framework also provides an extensible data plane that supports a variety of plugins, thereby enabling interoperability with a wide range of asset storage technologies. This ensures that data can be stored, accessed, and exchanged between different participants using different approaches.

The flexibility inherent in this framework is essential for accommodating the heterogeneous needs of participants within the dataspace. In this way, participants are able to use their preferred storage solutions without compromising interoperability.

The SEDIMARK EDC Connector [12] builds on this framework to enhance trustworthiness and decentralisation through the integration of DLT technologies and a smart contract architecture within the control plane. By integrating DLT, the connector ensures that all transactions are recorded on a decentralised ledger, enhancing transparency and reducing the risk of tampering or fraud. This decentralised approach not only improves security but also fosters greater trust among participants, as the immutable nature of the ledger provides a verifiable record of all offering-related transactions.

Once an agreement over a specific existing offering has been reached between two participants (a consumer and a provider) during the DSP-based negotiation phase, they interact with the smart contract architecture, in particular with the Fixed-Rate exchange smart contract. This smart contract facilitates the transfer of ownership of an ERC20 token to the consumer account. This token represents a signed agreement linked to the specific offering that was negotiated, which is represented by another NFT token, generated as the result of the offering tokenisation procedure during the offering registration stage. This process is illustrated in Figure 4. The smart contract ensures that the agreed-upon terms are executed securely and transparently by automating the transfer and recording it on the DLT, providing a verifiable and immutable record of the exchange.

In this particular context, the DLT Booth, as detailed in Figure 7, is the component responsible for facilitating interactions between the dataspace and DLT domains. Moreover, it acts as a bridge, offloading certain cryptographic operations and ensuring these processes are handled efficiently and securely. In this regard, it manages interactions with the Ethereum Virtual Machine (EVM), which is essential for executing smart contracts within the DLT environment. Furthermore, the DLT Booth also functions as a secure wallet, safeguarding the cryptographic credentials necessary for these operations. By securely storing and managing these credentials, it ensures that all transactions and interactions are protected against unauthorised access and potential security breaches. Overall, the introduction of the DLT Booth in the system simplifies the integration of DLT technology within the dataspace framework.

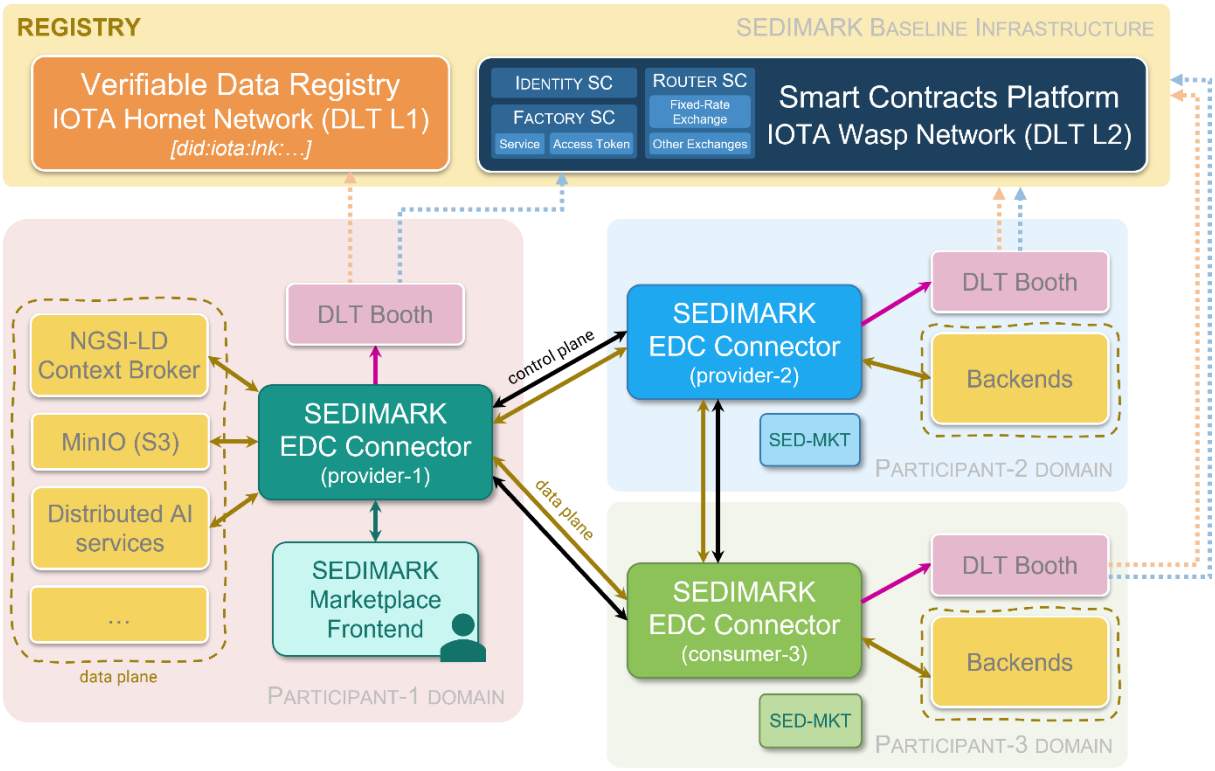


Figure 7 Asset exchange components

Once an agreement over an offering is reached, the offering procurement process, which ultimately leads to the actual asset exchange, can start. As introduced before, the asset exchange process relies on the data plane, a critical component that handles the actual transfer of data assets between participants. This transfer can follow different models, such as pull, push, or mixed. In the pull model, the consumer initiates the data transfer by requesting the asset from the provider. In contrast, the push model involves the provider initiating the transfer and sending the data asset to a destination indicated by the consumer. Finally, the mixed model combines elements of both pull and push approaches, allowing for more flexible and dynamic data exchange scenarios.

The data plane's capacity to support these different transfer models, along with its extensibility through the creation of new plugins to support additional technologies, ensures that the asset exchange process can be tailored to align with the specific requirements and preferences of the participants. This flexibility accommodates different types of assets and storage solutions. Examples of compatible technologies are, among others, REST APIs or AWS S3 API.

While the specific behaviour of each individual data plane plugin varies depending on the backend technology employed for asset transfer, they all share a common approach to authorisation. The previously acquired ERC20 token, obtained to represent an agreement on a specific offering, serves as the basis for authorisation purposes. Ownership of this token must be verified during the communication between participants to initiate the offering procurement process. Additional validations will be performed based on SEDIMARK Verifiable Credentials and the specific ODRL policies included in the offering agreement. Once authorisation is validated, technology-dependent interactions with the involved backend components will configure and control the asset exchange process. One example of such

interaction would be to use a storage API to generate compatible temporal access credentials that are then notified to the consumer.

3.1.2 Integration specification

Figures 8-9 present the current status of the tasks that need to be completed in order to integrate the MVM components. This board will be continuously updated to keep track of the integration progress.

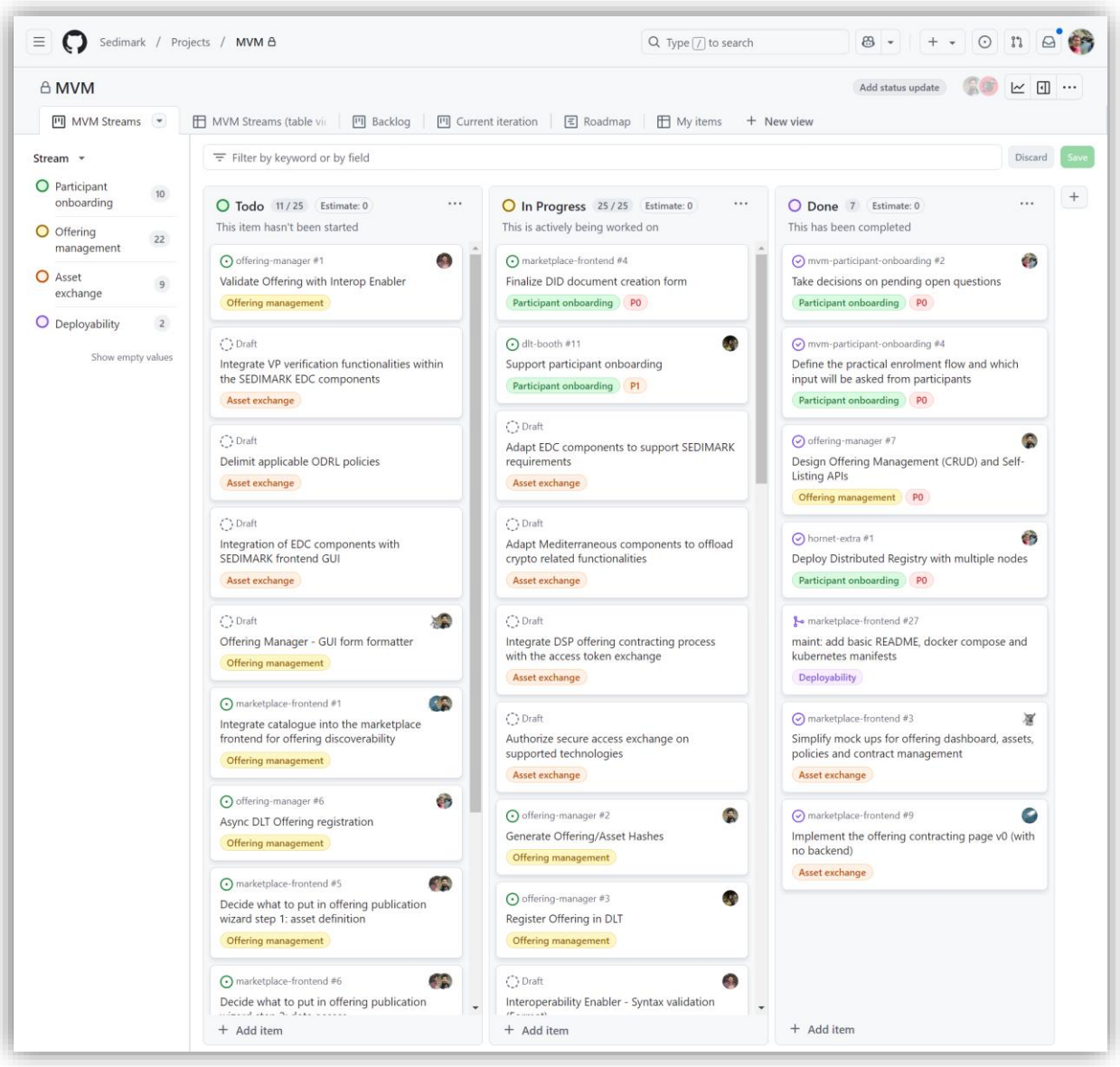


Figure 8 The MVM issue dashboard on the git repository (1)

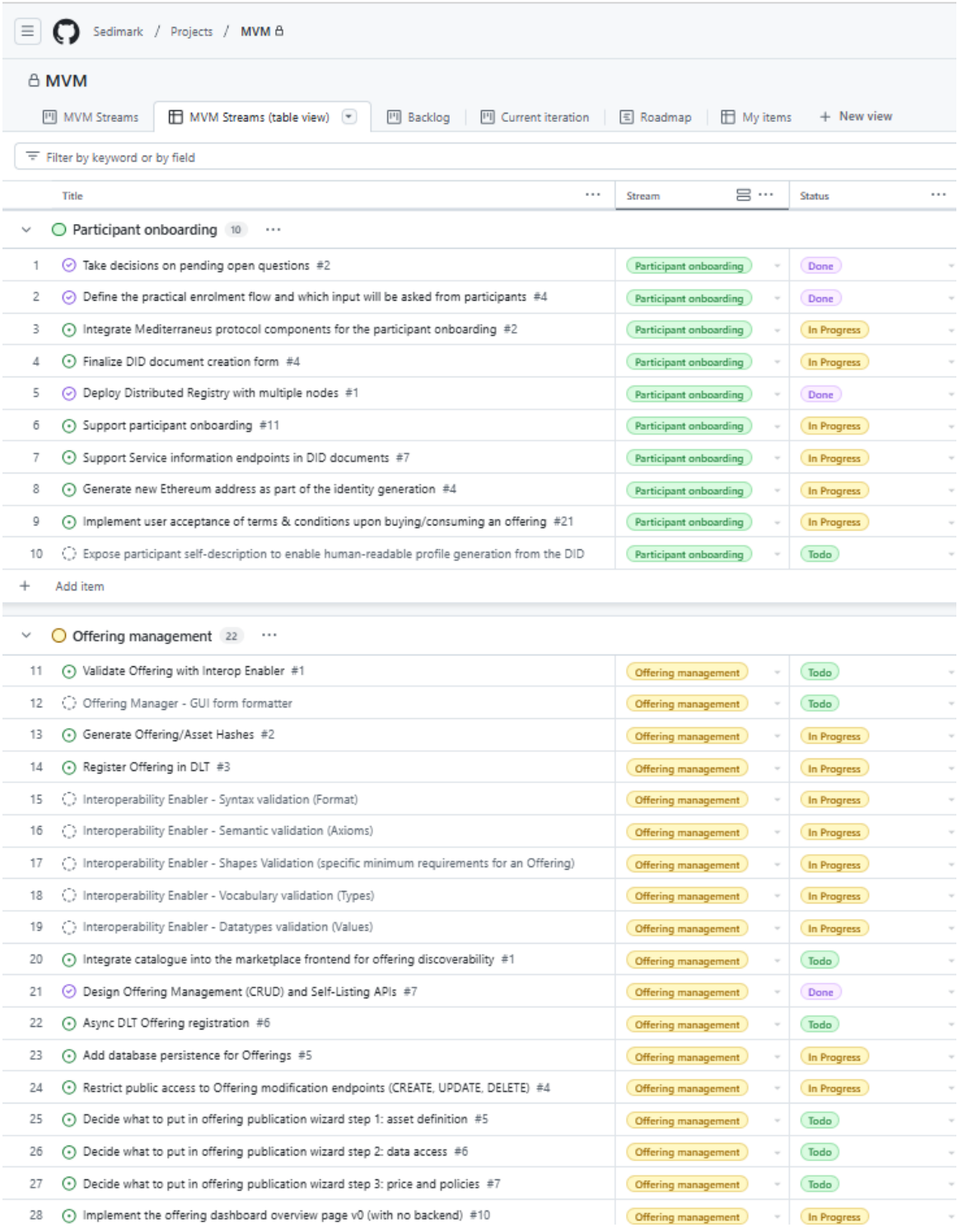


Figure 9 The MVM issue dashboard on the git repository (2)

The relevant link for the GitHub repository is: [18]

3.2 Minimum Viable Intelligence (MVI)

The Minimum Viable Intelligence (MVI) represent the core AI-driven functionalities of the SEDIMARK platform, enabling intelligent data processing capabilities, analytics, decision making and asset/data exchange within a decentralised framework. MVI bridges data providers and data consumers by leveraging advanced AI techniques, including federated learning, data curation, and distributed model training, to extract actionable insights while ensuring data privacy and security. By integrating AI orchestrators, data preprocessing pipelines, and distributed machine learning frameworks, MVI ensures the platform delivers robust intelligence capabilities and encourages participants to enhance data usability and unlock the full potential of the platform's decentralised marketplace. These features are particularly critical for supporting use cases such as data quality validation, predictive analytics, and event detection in real-time, making SEDIMARK a scalable and efficient data and services marketplace.

3.2.1 Define sub-streams and provide high-level description

The Minimum Viable Intelligence (MVI) contains the basic set of components required to facilitate the data processing pipeline management as well as the AI orchestration pipeline management within the SEDIMARK ecosystem. In Figure 10, we have presented from up to down, from the producer to the consumer (lower part of the image), how the involved components interact with each other.

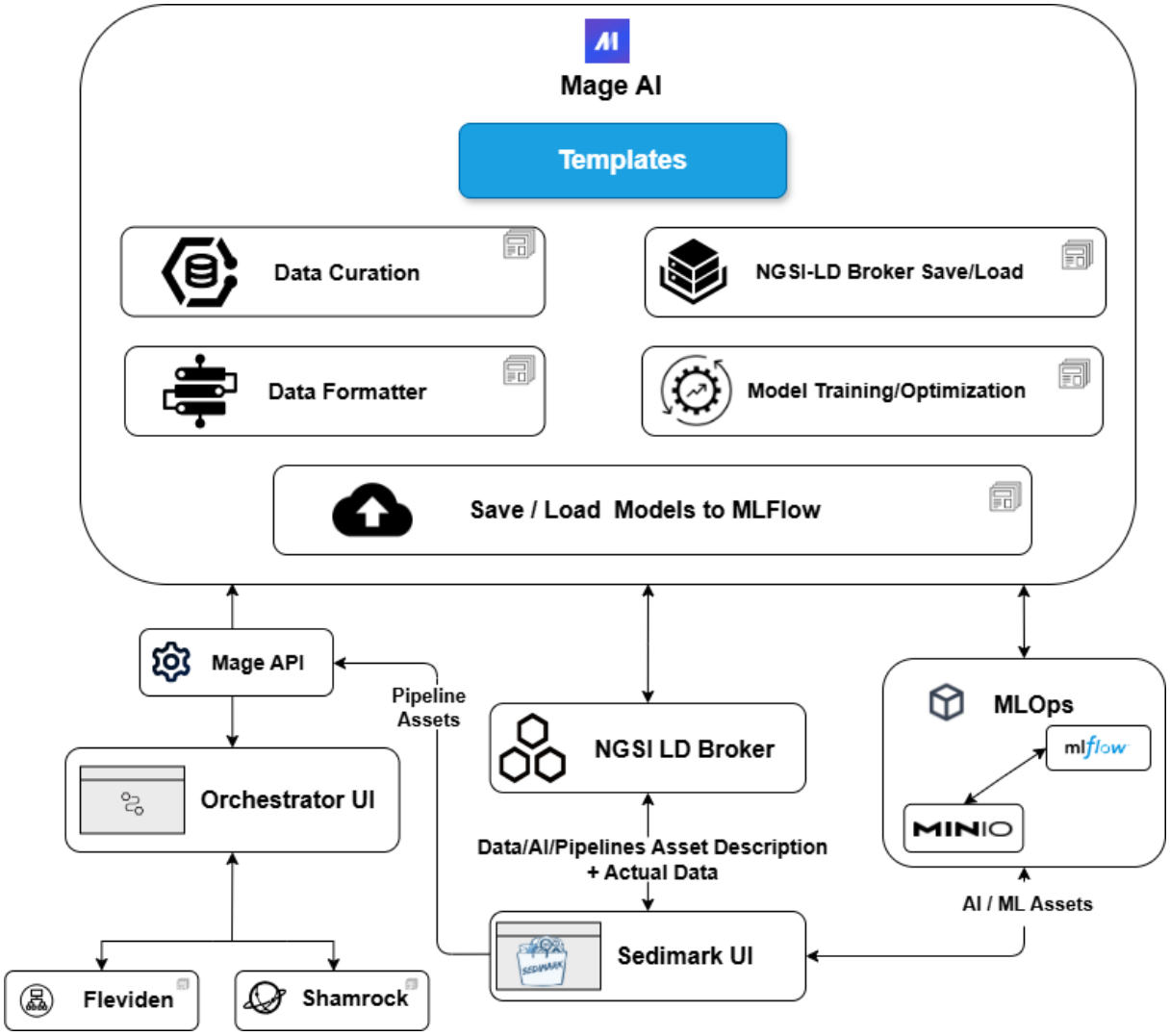


Figure 10 MVI Architecture

In what follows we will provide explanations for each component from the above diagram, focusing on orchestration of the AI models, data processing pipelines, and overall marketplace interactions.

Mage.AI is a framework that allows modelling of the transformation and integration of any data.

The modules that are integrated in the Mage.AI are:

- Data Curation which represents blocks that perform cleaning, profiling and deduplication of the input data.
- Data Formatter which transforms data from NGSI-LD, JSON to pandas' data frames, thus standardizing data formats.
- NGSI-LD Broker Save/Load. This block facilitates data reading and storage from and to the NGSI-LD broker. It also allows the user to select the entity where data will be stored to.
- Model Training/Optimisation supports the development and tunning of the parameters of the AI models for specific use cases.

The model management is realised with the help of MLFlow. MinIO serves as data storage for information resulting from training such as metrics, plots, confusion matrixes, etc.

The Orchestrator UI application connects users with pipelines defined in Mage.AI to set various parameters of the block or simply run them.

The same interface connects users with AI orchestrators which implies using tools like Shamrock or Fleviden.

The MVM includes a Global Catalogue that allows for organising asset descriptions (assets like data, AI models, and pipelines). These asset descriptions will enable the customer to discover the assets. Assets will be then made available to the customer via Connectors.

A Connector facilitates data exchange and interaction with NGSI-LD broker ensuring secure data flow between the Catalogue, Mage.AI and other MVM modules.

The NGSI-LD broker acts as a central hub for metadata, actual data and AI pipeline and asset descriptors.

Users can interact with the system through the orchestrator UI to manipulate existing pipelines.

3.2.1.1 Data storage

Set up MinIO for Data Storage and Access

The setup for MinIO is managed via a Docker Compose configuration file, which deploys MinIO alongside all other components included in the SEDIMARK MVI Toolbox. The specific configuration details for deploying MinIO will be outlined in Section 3.2.1.12.

MinIO will serve as a storage solution for models generated by AI pipelines executed in Mage, as well as for other artifacts associated with these models. While MinIO can also be utilised for storing data that cannot be directly saved to the NGSI-LD broker, this will not be its primary purpose.

MinIO will primarily be accessed through interactions between the components of the toolbox, triggered by specific flows initiated by the user. That said, direct access to MinIO is also possible using the credentials specified in the Docker Compose configuration, as shown in Figure 11. By using the ROOT credentials, the toolbox user can access the MinIO interface at http://localhost:9000 after deployment to view all stored data.

```
1  MINIO_ROOT_USER=admin
2  MINIO_ROOT_PASSWORD=minio
3  MINIO_ACCESS_KEY=default
4  MINIO_SECRET_KEY=default
```

Figure 11 MinIO credentials

MinIO is utilised within the platform to showcase the functionality of the marketplace, but it is not the sole or primary storage solution. Providers will have the flexibility to specify the source of their data, which can be stored using any storage method of their choice.

3.2.1.2 Data Processing

Integration between MageAI and Data Curation tools

Data Curation is a Python module developed as part of the SEDIMARK project to assist in cleaning and enriching data. A summary of the actions that can be performed using this module is outlined below:

- Anomaly detection/annotation
- Data deduplication
- Interpolation of missing data
- Data profiling

All these functions will be implemented as Mage AI template blocks that can be imported inside a pipeline based on the needs of a particular pipeline.

An example of a block for data curation tools can be seen in Figure 12:

```
    There should be one parameter for each output variable from each parent block.

    Args:
        data: The output from the upstream parent block
        args: The output from any additional upstream blocks (if applicable)

    Returns:
        Anything (e.g. data frame, dictionary, array, int, str, etc.)
    """

    valid_ranges = []
    numeric_columns = []
    categorical_columns = []

    for column in data.columns:
        if pd.api.types.is_numeric_dtype(data[column]):
            data[column] = data[column].apply(lambda x: 0.0 if np.isnan(x) else float(x))
            numeric_columns.append(column)

    data = data.dropna()
    data = DataSource(
        data,
        time_column="observedAt",
        valid_ranges=valid_ranges,
        categorical_columns=categorical_columns,
        numeric_columns=numeric_columns,
    )

    threshold_type = kwargs.get("threshold_type")

    config = {
            "model" : 'pyod_mcd',
            "processing_options":'describe',
            "model_config" : {
                'threshold_type':'AUCP',
            },
            "data_type":'tabular'
        }

    plt.rcParams['font.family'] = 'DejaVu Sans Mono'

    module = AnomalyDetectionModule(**config)
    result = module.process(data)
    df = result._df

    df['_is_anomaly'] = df['_is_anomaly'].apply(lambda x: 0 if not x else 1)
    df['_anomaly_score'] = df['_anomaly_score'].apply(lambda x: float(round(x, 2)))

```

Figure 12 Data Curation block

Integration between MageAI and Data Formatter

Data Formatter is a collection of Python modules and scripts designed to standardize data formats both during the execution of a pipeline in Mage AI and at its conclusion, ensuring consistent data storage. To achieve this, NGSI-LD was selected as the primary format for saving and retaining most, if not all, data for future use within the SEDIMARK Marketplace.

However, the Data Formatter can also support CSV and XLS/XLSX formats, enabling flexibility when dealing with data in various formats.

The integration between Mage AI and the Data Formatter will be achieved through template blocks created in Mage. These blocks will be incorporated into data processing pipelines to handle data conversion and formatting into NGSI-LD, either at the beginning or the end of the pipeline.

Data Quality Annotation is performed after the data processing pipelines and is designed to add quality annotations directly to the output (i.e., pandas DataFrame) of the pipeline. These annotations can be applied either at the attribute level (to specific columns or features) or the instance level (to entire rows).

To transform the enriched Dataframe into the NGSI-LD format, the Data Mapper is utilised.

The integration between Mage AI and these two components will be achieved through the use of template blocks designed within Mage.

Regarding the AI pipeline, two new components are currently in progress to enable the dynamic transformation and restoration of DataFrames. The first component (Data Transformation) is performed after the Data Formatter and is designed to extract relevant data from the NGSI-LD data in the Broker, creating a focused subset (DataFrame) specifically for AI model training. The second one (Data Restoration) ensures the original structure is restored by mapping the prediction back to the full data, maintaining consistency and coherence. Finally, an NGSI-LD output is generated thanks to the Data Mapper and can be stored back to the Broker.

All these components will be integrated in Mage AI.

Integration between MageAI and Context broker for loading/saving

The context broker in the case of the MVI architecture will be the NGSI-LD broker, this broker will be used to standardize the data format inside the SEDIMARK Marketplace.

The broker, like all other components in the toolbox, will be deployed as a Docker container. Interaction between Mage and the Context Broker will occur at the end of a pipeline via template blocks. These blocks will first collect all the information related to a specific asset. In SEDIMARK, assets can be categorised as data assets, AI/ML model assets, or pipeline assets. The collected information will then be converted into NGSI-LD format and saved in the Context Broker. If the asset is a data asset, the associated data will also be stored in the broker. Figure 13 presents the flow of data, starting from Mage and going until the context broker.

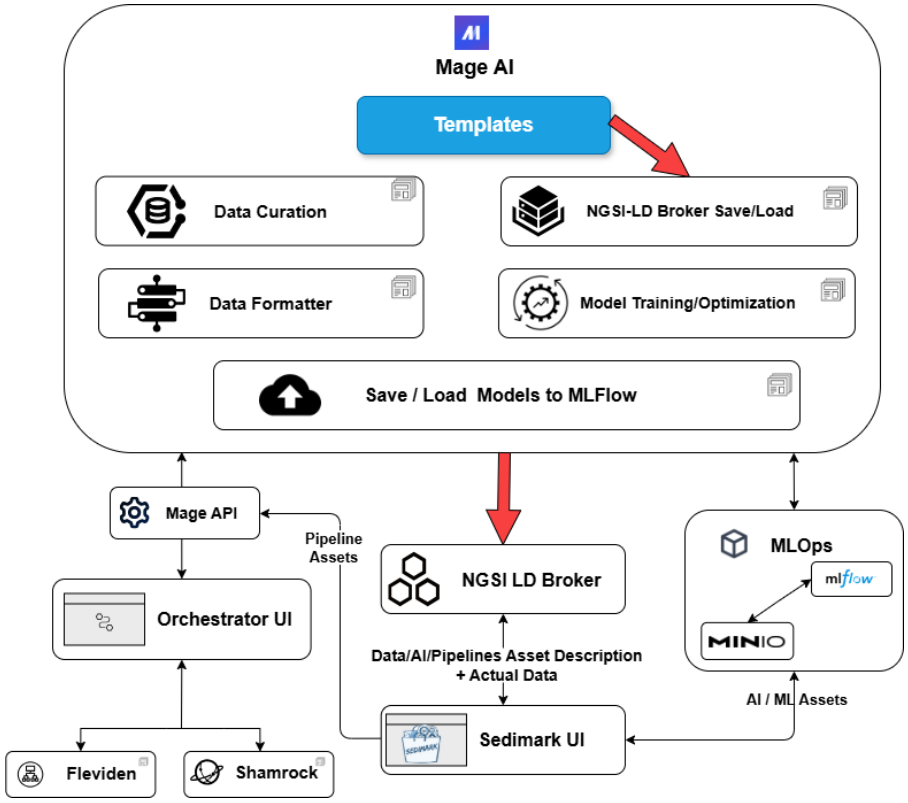


Figure 13 Context Broker saving flow

3.2.1.3 AI Orchestrator

The diagram in Figure 14 represents the flow of a Federated Learning process between an AI Enabler Initiator and an AI Enabler Participant. This process is similar both in the Shamrock tool as well as in the Fleviden tool which will be described in what follows.

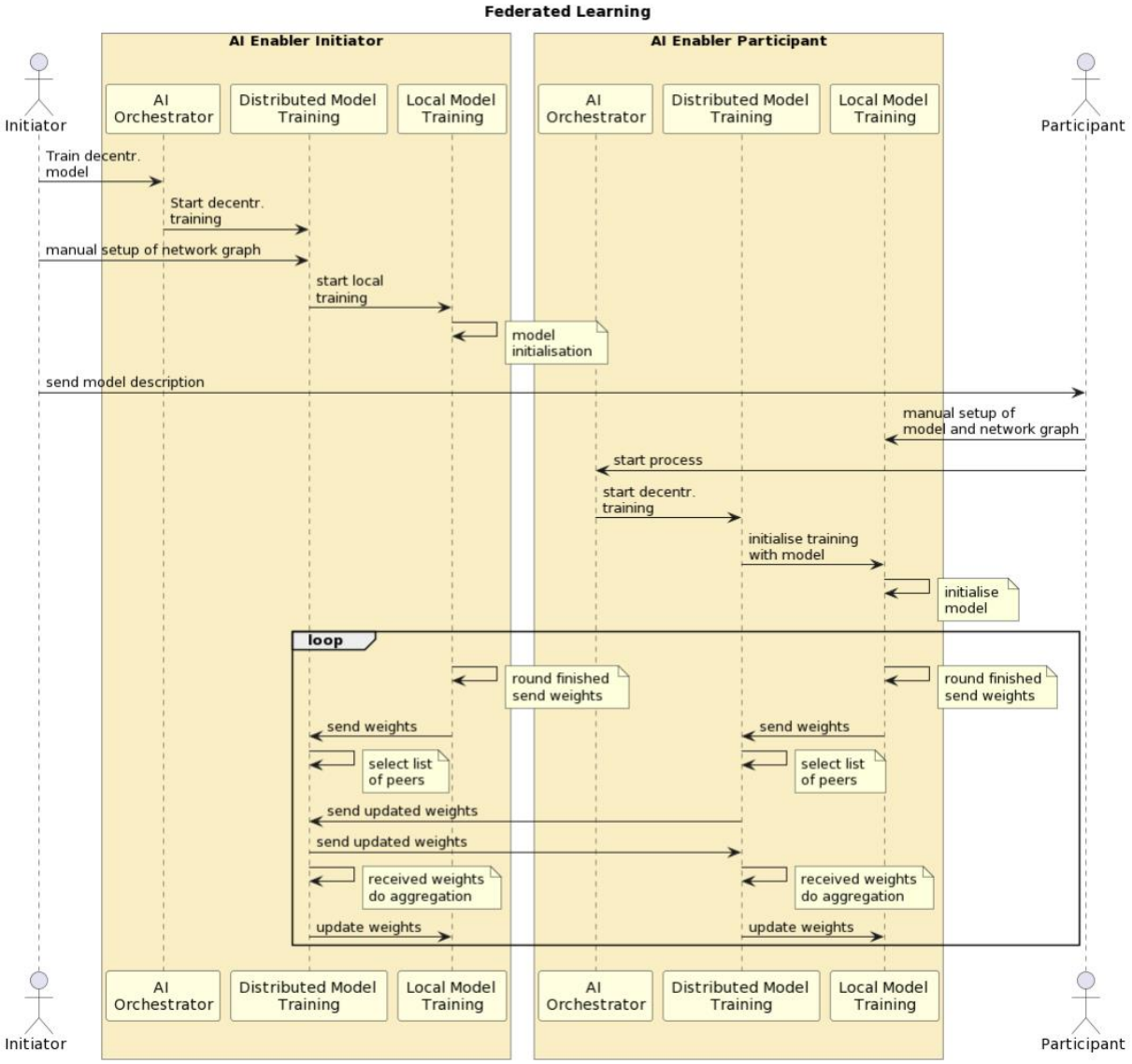


Figure 14 Federated learning block

The federated learning block comprises three main steps:

- Model initialisation
  - The initiator begins training a decentralised model by setting manually the network graph for participating nodes.
  - The model description is sent to the participant who also performs a manual setup of its local model.
- Training initialisation
  - Both will initialize the model and start the distributed process.
  - Local training occurs on each node.
- Iterative loop
  - Each participant sends its local weights after a round of training.

- The AI Orchestrator collects the weights from all participants, aggregates them and updates the global model.
- The updated model weights are sent back to each participant who incorporates them in their models to start the next training round.
- The process iterates for multiple rounds until the training converges.

Integration between AI Orchestrator and Shamrock

SEDIMARK provides Shamrock, a lightweight and composable tool for enabling distributed training of machine learning models. Shamrock is based on a simple node architecture, where nodes host datasets and machine learning models, and can be composed via the definition of a topology to train local models while communicating and aggregating weights. In its current implementation, Shamrock provides support for both Keras and PyTorch models, with Keras allowing for training to take place in two additional backends - Jax and Tensorflow- provided the model is defined using Keras syntax. Shamrock will feature additional support for both split learning and federated distillation. Shamrock nodes may communicate both model weights, as well as fully defined models in either Pytorch of Keras, and additional metadata, such as sharing lists of peers in order to expand decentralised networks. Shamrock operates primarily over a REST http API, with nodes operating Starlette servers, and features one-way request-response communication.

Integration between AI orchestrator and Fleviden

SEDIMARK also provides Fleviden as an alternative framework for decentralised federated learning. Fleviden follows a pipes-and-filters paradigm that allows users to define a highly flexible computational graph by instantiating and connecting Fleviden pods. This flexibility is what enables different training topologies, such as client-server, hierarchical or swarm learning, as well as more complex custom scenarios. Currently, Fleviden supports Keras, Pytorch and Scikit-Learn for training the models, and HTTP or Kafka-based communication between agents. Additionally, Fleviden offers privacy-preserving mechanisms, model performance monitoring through MLFlow, and several compression and quantisation techniques to reduce the energy consumption of the process.

Integration between AI orchestrator and model optimisation modules

The interaction between the AI orchestrator and the model optimisation modules begins with an AI pipeline created in Mage, utilizing various block templates that contain code from the model optimisation modules. Through configurable parameters, the optimisation process can be fine-tuned for a specific dataset and model.

At the end of the AI pipeline, a dedicated block will save the asset description of the resulting model into the context broker and store the model itself in the AI orchestrator. In the MVI architecture, the AI orchestrator is represented by the MLFlow and MinIO components. The integration workflow from the AI pipeline to the AI orchestrator is illustrated in Figure 15.

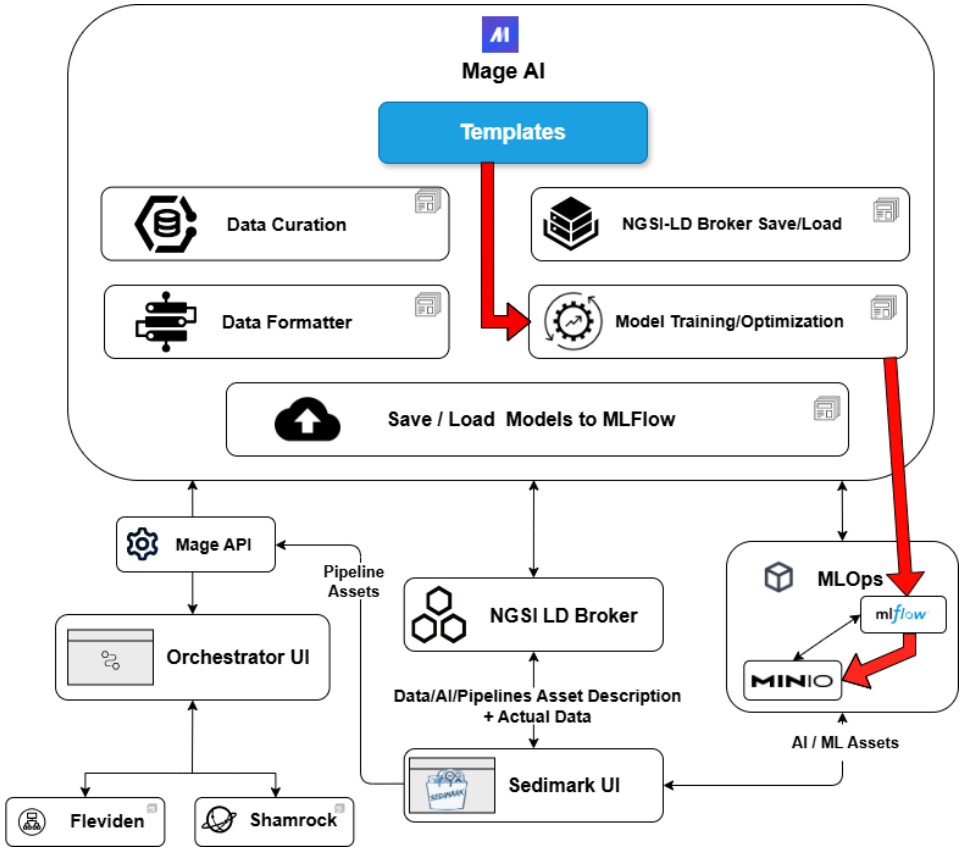


Figure 15 Integration flow between AI orchestrator and model optimisation modules

3.2.1.4 Local Model Training

Local model training automatisation and interface design

The design of interfaces for local model training enables participants to interact with the training pipeline via both GUIs and APIs….

The GUI provides a user-friendly interface for initiating and monitoring local model training tasks. Key features include:

- Dataset selection: The user retrieves a dataset from the marketplace using file upload mechanisms (drag and drop) to choose datasets. An indicative interface is shown in Figure 16.

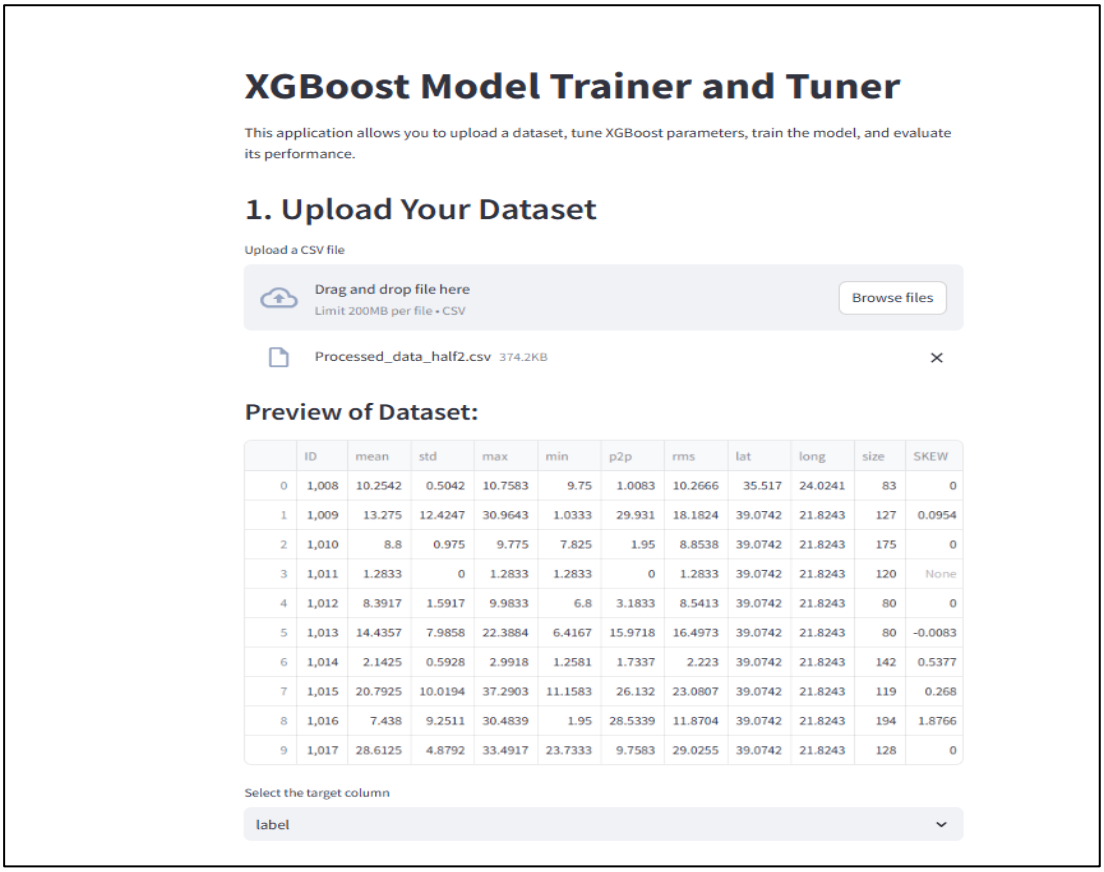


Figure 16 Interface for uploading the dataset to train and evaluate the model.

- Model configuration: The user chooses from a series of interactive forms for selecting model types, defining hyperparameters, and choosing training schedules (Figure 17).

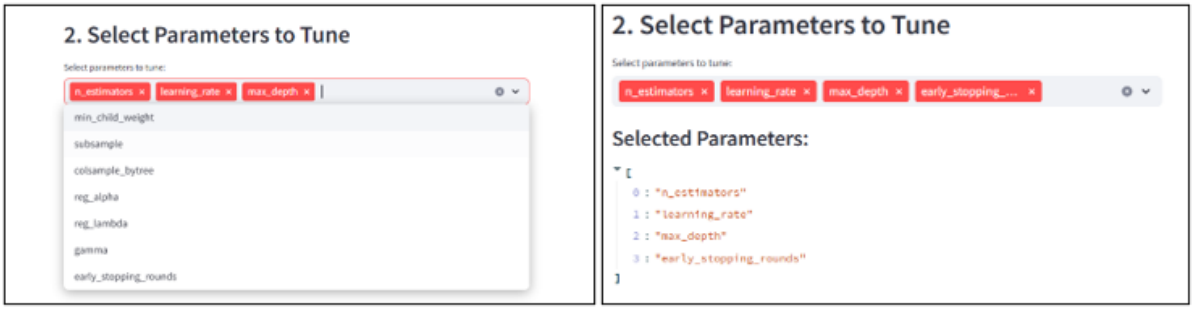


Figure 17 Options for selecting the hyperparameters of the XGBoost Regressor model

- Parameter selection and model tuning: Users have the option to set the parameters for the XGB model. Then, they can click a “Tune and Train” button to perform hyperparameter tuning and train the model with the best parameters (Figure 18).

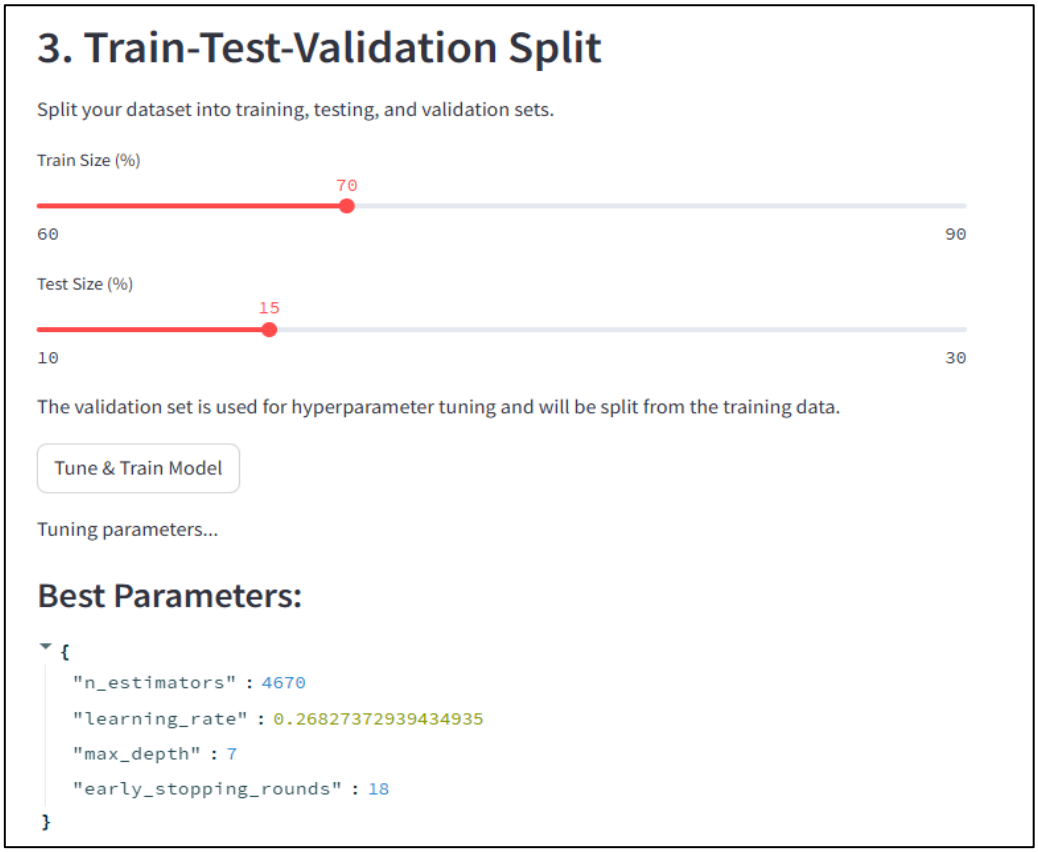
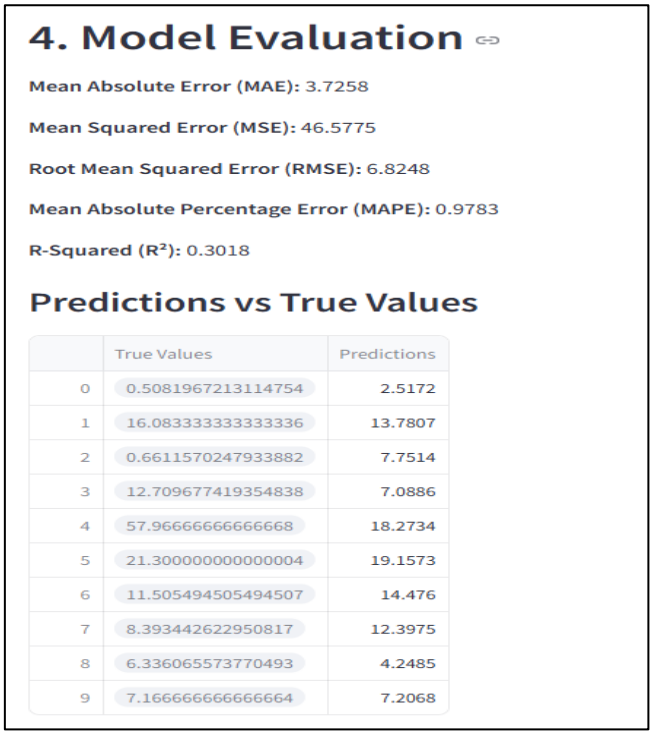


Figure 18 User interface for selecting the train-test split percentages for the uploaded dataset.

- Model evaluation and results visualisation: After training, the UI displays model performance metrics and evaluation results. Users can view graphical comparisons, such as predictions vs. true values, to analyse the model's accuracy and mean daily consumption predictions. The following dashboards illustrate how the chain of the results is displayed on the interface (Figure 19-22).



4. Model Evaluation

Mean Absolute Error (MAE): 3.7258

Mean Squared Error (MSE): 46.5775

Root Mean Squared Error (RMSE): 6.8248

Mean Absolute Percentage Error (MAPE): 0.9783

R-Squared (R^2): 0.3018

Predictions vs True Values

	True Values	Predictions
0	0.5081967213114754	2.5172
1	16.083333333333336	13.7807
2	0.6611570247933882	7.7514
3	12.709677419354838	7.0886
4	57.96666666666668	18.2734
5	21.300000000000004	19.1573
6	11.505494505494507	14.476
7	8.393442622950817	12.3975
8	6.336065573770493	4.2485
9	7.166666666666664	7.2068

Figure 19 Table displaying a comparison of the predicted values and the true values for energy consumption.

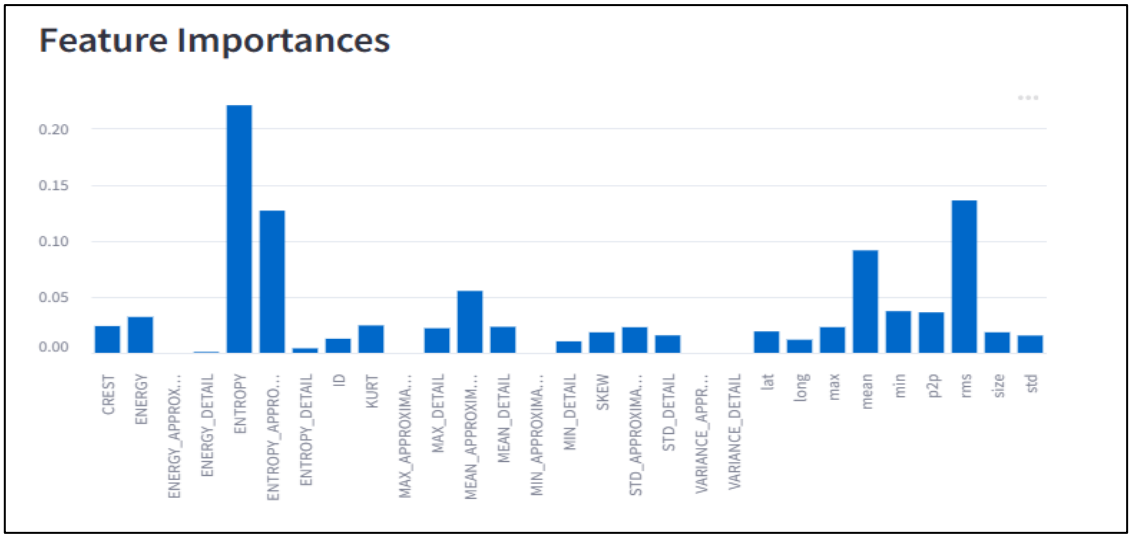


Figure 20 Visualisation of feature importances from the dataset

Analysis

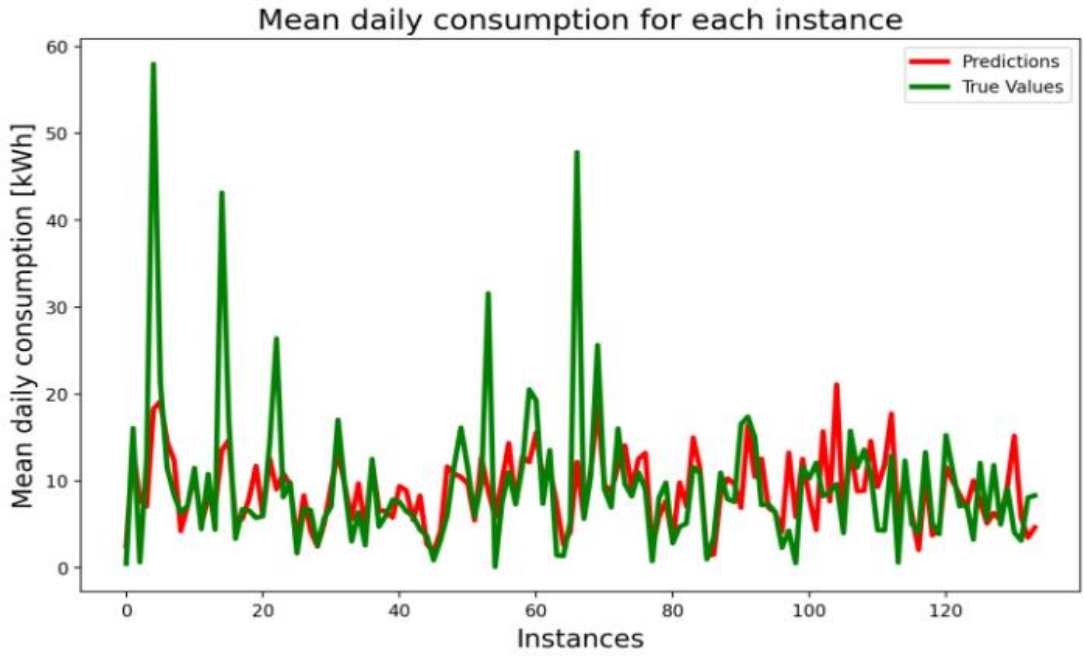


Figure 21 Line plot comparing the predicted vs. true values of mean daily energy consumption for each instance in the test set.

The graph shows that the model captures the general trend of energy consumption, particularly for low-to-moderate values. While the model slightly underpredicts peaks, it performs well in aligning with the overall behaviour of the data.

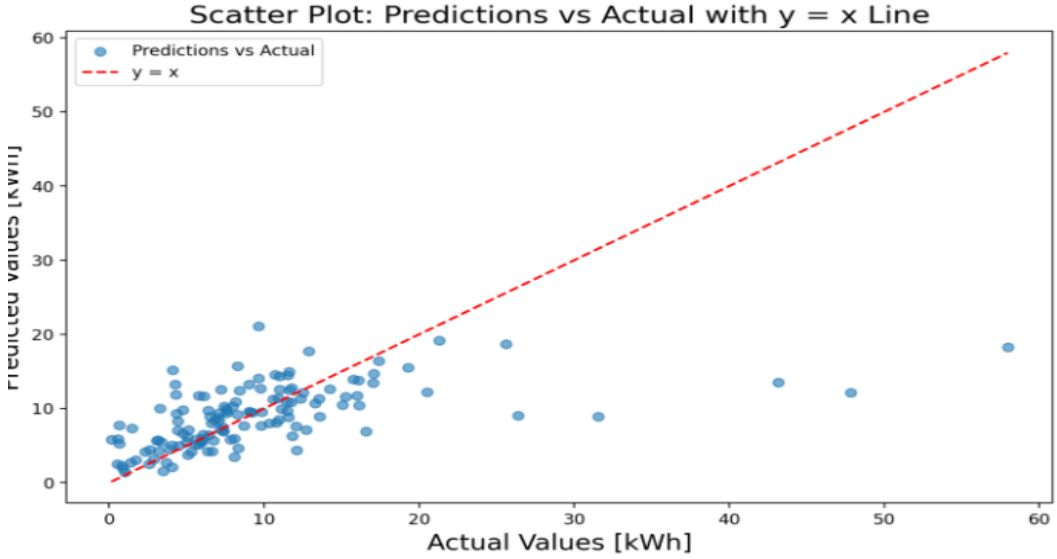


Figure 22 Scatter plot of predictions vs. true values

The above scatter plot highlights that the model performs reliably for lower energy values where predictions align closely with the actual values. Although deviations increase for higher

values, the overall pattern follows the expected trend, indicating the model's potential with further refinement.

The purpose of the whole analysis is to help the user assess how closely the model's predictions align with the actual values, identify areas where the model might overpredict or underpredict energy consumption, and provide a visual overview of the model's accuracy and performance across the entire test set. More details can be found in the GitHub project repository here: [20]

3.2.1.5 Distributed Model Training

Toolbox Deployment for Shamrock

Shamrock is a distributed machine learning tool available through the SEDIMARK toolbox. Shamrock is deployed as a Python package, which is subsequently imported by MageAI. Users can then use the functionality of Shamrock through the MageAI pipeline. In particular, users will be able to select a particular Keras or Pytorch model to train in the distributed settings. Moreover, users can choose from several distributed learning topologies and the specific model part for communication (i.e. model weights or model description). An example of running a decentralised model training in the federated learning topology is shown in Section 3.2.1.3.

Toolbox deployment for Fleviden

Fleviden is another federated learning tool that SEDIMARK toolbox also developed as a Python framework. After installing the Fleviden library, users will be able to define and customize their federated learning scenario, for example, defining the topology (Client-Server, Hierarchical, Swarm Learning, etc), the training framework (Keras, Pytorch, Sklearn), the communication protocol (HTTP, Kafka), the kind of data the model will process (tabular, image, etc.), and additional features (compression techniques, privacy preservation, client-selection, etc). Clients will be able to subscribe and unsubscribe dynamically to a federated learning network, being able to join already initiated training processes.

3.2.1.6 Model Management

Implement MLFlow for Experiment Tracking and Model Management

MLFlow is an open-source platform designed to streamline the management of machine learning (ML) workflows. It provides tools for tracking experiments, packaging models, and managing the deployment lifecycle, and is used to maintain reproducibility, collaboration, and scalability in AI/ML projects.

MLFlow offers robust features for managing the machine learning lifecycle. It provides experiment tracking by logging and visualizing metrics, parameters, and artifacts like models and datasets, enabling efficient comparison and record-keeping. A centralised model registry ensures versioning, metadata management, and stage transitions, with the ability to roll back to previous versions.

MLFlow standardizes model packaging, allowing seamless deployment across environments such as REST APIs, batch pipelines, embedded systems, or cloud platforms like AWS SageMaker, Azure ML, and Kubernetes. It integrates smoothly with popular ML libraries such as TensorFlow, PyTorch, XGBoost, and Scikit-Learn, as well as major data platforms like Databricks and Apache Spark. With support for multiple programming languages, including Python, R, and Java, MLFlow accommodates diverse technical teams effectively.

MLFlow is deployed inside the SEDIMARK toolbox as a Docker container, to provide a standardised way for the components deployed and ensure seamless communication between the different components. Figure 23 presents the user interface of MLFlow, which can be used by a user inside the SEDIMARK platform to see how the different models are stored and to visualize the metadata around them, but communication with MLFlow will be done mostly through code and API calls.

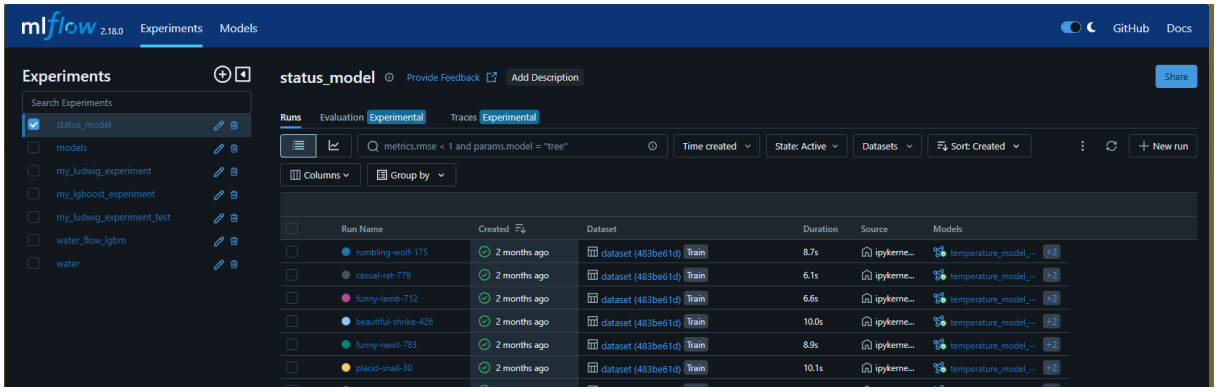


Figure 23 MLFlow UI

Integrate MLFlow with MageAI for Model Management

The integration between MLFlow and Mage AI will be done as for most of the other components inside the MVI toolbox, through template blocks that will contain code for the following operations:

- Storing trained models
- Getting trained models from storage to use for predictions
- Updating a model

These three operations will ensure seamless integration between Mage and MLFlow. The template blocks will be parameterised to provide flexibility, accommodating various models generated through different AI pipelines.

PY TRANSFORMER train_hydro_series ←O 1 parent

```
mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100



metrics={
        "mse":mse,
        "mse_scaled":mse_scaled,
        "rmse":rmse,
        "rmse_scaled":rmse_scaled,

        "mae":mae,
        "r_squared":r_squared,
        "mape":mape,
        "mean_precision":mean_precision
        }


y_test.reset_index(drop=True,inplace=True)


df_compare=pd.DataFrame()
df_compare['y_pred']=y_pred
df_compare['y_test']=y_test
print(f"df_compare {df_compare}")

# Infer the model signature-MLFLOW
signature = infer_signature(X_test, y_pred)

plot_predictions(test_data,y_test,y_pred)
plot_real_pred(y_test,y_pred)

client = MlflowClient()

# client.create_registered_model("water_model",tags={"model_type": "LGBM", "mage_model": "true"})

model_name = "water_model"
try:
    registered_model = client.get_registered_model(model_name)
except Exception as e:
    registered_model = client.create_registered_model(model_name, tags={"model_type": "LGBM", "mage_model": "true"})


with mlflow.start_run(experiment_id=mlflow.get_experiment_by_name("water").experiment_id) as run:
    mlflow.sklearn.log_model(
        sk_model=best_model,
        # artifact_path="water_flow_model",
        artifact_path="water_model",

        registered_model_name="water_model",
        signature=signature,

    )

    mlflow.log_artifact("water_flow_model.png", artifact_path="figures")
    mlflow.log_artifact("water_flow_model_vs.png", artifact_path="figures")
```

Figure 24 MLFlow storing block

```python
@data_exporter
def export_data(data, *args, **kwargs):
    """
    Exports data to some source.

    Args:
        data: The output from the upstream parent block
        args: The output from any additional upstream blocks (if applicable)

    Output (optional):
        Optionally return any object and it'll be logged and
        displayed when inspecting the block run.
    """

    model_name = data[0]
    print(f"model name :{model_name}")

    X_test,ytest,model_name=linear_regression_predict(data,model_name)

#

    run_id="3542d79b08d14161bbace64050007a01"
    logged_model = f'runs:/{run_id}/{model_name}'


    print(logged_model)

    # loaded_model = mlflow.pyfunc.load_model(logged_model)
    loaded_model = mlflow.sklearn.load_model(logged_model)
    print(loaded_model)
    # # loaded_model = mlflow.sklearn.load_model(f"models:/{model_name}/1")
    predictions = loaded_model.predict(X_test)


    return X_test,ytest,predictions
```

Figure 25 MLFlow loading and prediction block

Figure 24 illustrates a block for storing a trained model in MLFlow, while Figure 25 demonstrates a block for loading a trained model and performing predictions on a given dataset.

Integrate MageAI with energy consumption data

The integration of MageAI with energy consumption data enables advanced data analytics and machine learning workflows, particularly for time-series forecasting and energy demand prediction. This integration brings automated machine learning (AutoML) capabilities into the platform, simplifying the process of building, training, and deploying predictive models for energy consumption patterns while maintaining privacy and data locality.

The data processing and model training pipeline that we have designed and implemented consists of four sequential blocks:

- Data Loader block: Fetches and integrates the datasets from a GitHub repository associated with the project. The processed dataset is saved for downstream tasks (Figures 26-27).

```python
@data_loader
def load_data_from_api(*args, **kwargs):
    """
    Template for loading data from API
    """
    url = 'https://raw.githubusercontent.com/Sedimark/local_model_training/main/Data/Processed_data_half2.csv'

    # Retrieve the token from environment variable
    token = get_secret_value('github_token')

    # Add token to headers for authentication
    headers = {'Authorization': f'token {token}'}

    # Make the GET request with headers
    response = requests.get(url, headers=headers)

    # Raise an error if the request failed
    response.raise_for_status()

    df = pd.read_csv(io.StringIO(response.text), sep=',')

    # Read and return the CSV data
    return df
```

Figure 26 MageAI Data Loader block

	ID	mean	std	max	min	p2p	rms	lat
0	1008	10.2541666667	0.5041666667	10.7583333333	9.75	1.0083333333	10.2665533679	35.5169936
1	1009	13.275	12.4247494952	30.9642857143	1.0333333333	29.930952381	18.1824097693	39.074208
2	1010	8.8	0.975	9.775	7.825	1.95	8.8538480335	39.074208
3	1011	1.2833333333	2.220446049e-16	1.2833333333	1.2833333333	0	1.2833333333	39.074208
4	1012	8.3916666667	1.5916666667	9.9833333333	6.8	3.1833333333	8.5412804791	39.074208
5	1013	14.4356846473	7.9858127946	22.3884297521	6.4166666667	15.9717630854	16.4973390953	39.074208
6	1014	2.1424936387	0.592810479	2.9918032787	1.2580645161	1.7337387626	2.2229942096	39.074208
7	1015	20.7925311203	10.0193641499	37.2903225806	11.1583333333	26.1319892473	23.0806630832	39.074208
8	1016	7.4380165289	9.2510682614	30.4838709677	1.95	28.5338709677	11.8703982183	39.074208
9	1017	28.6125	4.8791666667	33.4916666667	23.7333333333	9.7583333333	29.0255305483	39.074208

893 rows x 30 columns

Figure 27 Energy consumption dataset features

- Data Splitter block: The dataset is split into training, test, and validation sets (Figure 28).

```python
@custom
def splitto_train_test(df, *args, **kwargs):

    # Splitting the DataFrame into features (X) and labels (y)
    X = df.iloc[:, 1:-1].values
    y = df.iloc[:, -1].values.reshape(-1, 1)

    # Train, validation, and test split
    train_len = 0.7
    X_train, X_temp, y_train, y_temp = train_test_split(X, y, train_size=train_len, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

    # Convert NumPy arrays to lists for compatibility with Mage AI
    result = {
        "X_train": X_train.tolist(),
        "X_val": X_val.tolist(),
        "X_test": X_test.tolist(),
        "y_train": y_train.tolist(),
        "y_val": y_val.tolist(),
        "y_test": y_test.tolist()
    }

    return result
```

Figure 28 Process of splitting the dataset using MageAI Data Splitter block

- Data Scale block: Applies standard scaler to normalise the dataset (Figure 29). The basic model we use is the **XGBoostRegressor** model.

```python
@custom
def scaler(data, *args, **kwargs):
    """
    Scales the train, validation, and test sets using StandardScaler.
    Accepts a dictionary as input, with keys 'X_train', 'X_val', 'X_test', 'y_train', 'y_val', and 'y_test'.
    Returns a dictionary with scaled datasets.
    """


    X_train = np.array(data['X_train'])
    X_val = np.array(data['X_val'])
    X_test = np.array(data['X_test'])
    y_train = np.array(data['y_train'])
    y_val = np.array(data['y_val'])
    y_test = np.array(data['y_test'])

    # Initialize scalers
    scaler_X = StandardScaler()
    scaler_y = StandardScaler()

    # Scale features (X)
    X_train_scaled = scaler_X.fit_transform(X_train)
    X_val_scaled = scaler_X.transform(X_val)
    X_test_scaled = scaler_X.transform(X_test)

    # Scale targets (y)
    y_train_scaled = scaler_y.fit_transform(y_train)
    y_val_scaled = scaler_y.transform(y_val)
    y_test_scaled = scaler_y.transform(y_test)

    # Return scaled data as lists for Mage AI compatibility
    return {
        "X_train_scaled": X_train_scaled.tolist(),
        "X_val_scaled": X_val_scaled.tolist(),
        "X_test_scaled": X_test_scaled.tolist(),
        "y_train_scaled": y_train_scaled.tolist(),
        "y_val_scaled": y_val_scaled.tolist(),
        "y_test_scaled": y_test_scaled.tolist()
    }
```

Figure 29 Scaling using the Data Scale block of MageAI

Integrate MLFlow with the energy prediction model

MLFlow integration within the SEDIMARK platform is critical for managing the lifecycle of ML prediction models. This integration allows the platform to streamline the process of model tracking, versioning, and deployment while ensuring the reproducibility and scalability of predictive analytics workflows.

More specifically, the integration leverages the continuation of the previous pipelines with a last Mage.AI block, namely the model block. We implemented the following processes:

- **Hyperparameter Tuning and Model Training**: The XGBoost model was tuned to identify the best-performing parameters and used them to train the model.
- **Model Deployment**: The trained model was uploaded to the GitHub repository here: [20]. The model is also saved in a MinIO bucket for storage and further usage.

```python
# Define the objective function for optuna tuning
def objective(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 5000),
        'learning_rate': trial.suggest_float('learning_rate', 1e-5, 0.3),
        'max_depth': trial.suggest_int('max_depth', 3, 15),
        'early_stopping_rounds': trial.suggest_int('early_stopping_rounds', 10, 50)
    }

    model = xgb.XGBRegressor(
        n_estimators=params['n_estimators'],
        learning_rate=params['learning_rate'],
        max_depth=params['max_depth'],
        early_stopping_rounds=params['early_stopping_rounds'],
        n_jobs=-1
    )

    # Train the model with early stopping and pruning
    model.fit(
        X_train,
        y_train,
        eval_set=[(X_val, y_val)],
        verbose=False
    )
    # Predict on validation set
    predictions = model.predict(X_val)

    # Minimize Mean Absolute Percentage Error
    return mean_absolute_percentage_error(y_val, predictions)
```

Figure 30 XGBoost model tuning/training using MLFlow

```
    with mlflow.start_run(experiment_id=current_experiment.experiment_id):

        final_xgb_model = xgb.XGBRegressor(
        n_estimators=best_params['n_estimators'],
        learning_rate=best_params['learning_rate'],
        max_depth=best_params['max_depth'],
        early_stopping_rounds=best_params['early_stopping_rounds'],
        n_jobs=-1
        )
        final_xgb_model.fit(
            X_train, y_train,
            eval_set=[(X_val, y_val)],
            verbose=False
        )

        # Make predictions
        y_train_pred = final_xgb_model.predict(X_train)
        y_val_pred = final_xgb_model.predict(X_val)

        # Calculate metrics for train set
        train_mse = mean_squared_error(y_train, y_train_pred)
        train_rmse = root_mean_squared_error(y_train, y_train_pred)
        train_r2 = r2_score(y_train, y_train_pred)
        train_mae = mean_absolute_error(y_train, y_train_pred)
        train_mape = (abs((y_train - y_val_pred) / y_train).mean()) * 100

        # Log the calculated metrics to MLflow
        mlflow.log_metric("Train MSE", train_mse)
        mlflow.log_metric("Train RMSE", train_rmse)
        mlflow.log_metric("Train R2", train_r2)
        mlflow.log_metric("Train MAE", train_mae)

        # Calculate metrics for validation set
        test_mse = mean_squared_error(y_val, y_val_pred)
        test_rmse = root_mean_squared_error(y_val, y_val_pred)
        test_r2 = r2_score(y_val, y_val_pred)
        test_mae = mean_absolute_error(y_val, y_val_pred)
        test_mape = (abs((y_val - y_val_pred) / y_val).mean()) * 100

        # Log the calculated metrics to MLflow
        mlflow.log_metric("Validation MSE", test_mse)
        mlflow.log_metric("Validation RMSE", test_rmse)
        mlflow.log_metric("Validation R2", test_r2)
        mlflow.log_metric("Validation MAE", test_mae)
        mlflow.log_metric("Validation MAPE", test_mape)

        mlflow.xgboost.log_model(final_xgb_model, artifact_path="model")


# Return the model, metrics, and best parameters
    return {
        'best_params': best_params,
        'metrics': {
            'rmse': test_rmse,
            'mae': test_mae,
            'mape': test_mape,
            'r2': test_r2
        }
    }
```

Figure 31 MLFlow code for XGBoost model

After running the code, the results are stored in the MLFlow as shown in Figures 32-33.

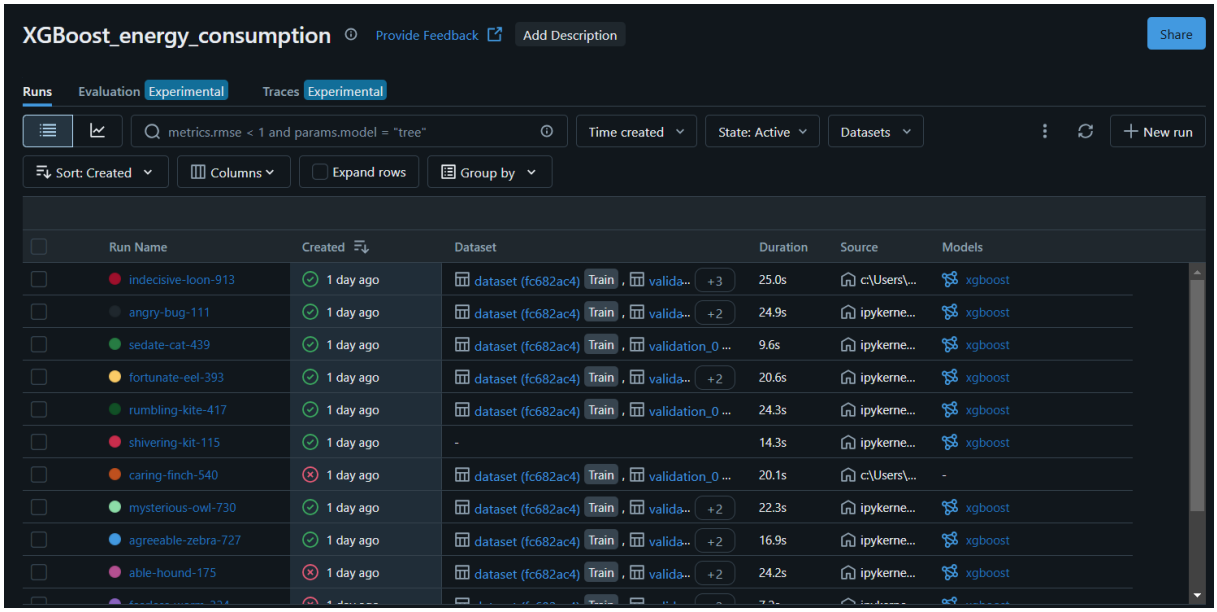


Figure 32 Stored model, dataset, metrics and artifacts in MLFlow (1)

Figure 33 Stored model, dataset, metrics and artifacts in MLFlow (2)

3.2.1.7 Asset Description

As part of the Asset generation process through the Orchestrator, Asset Descriptions are created as a by-product of the Asset itself. The Asset Description is used by Offering Manager Clients (UI or API) for creating the Offering Descriptions that will be published in the Marketplace Catalogue. The Connector will also use it to locate where the Asset is stored for retrieval and exchange with the Consumer. The above steps are depicted in Figures 34-36.

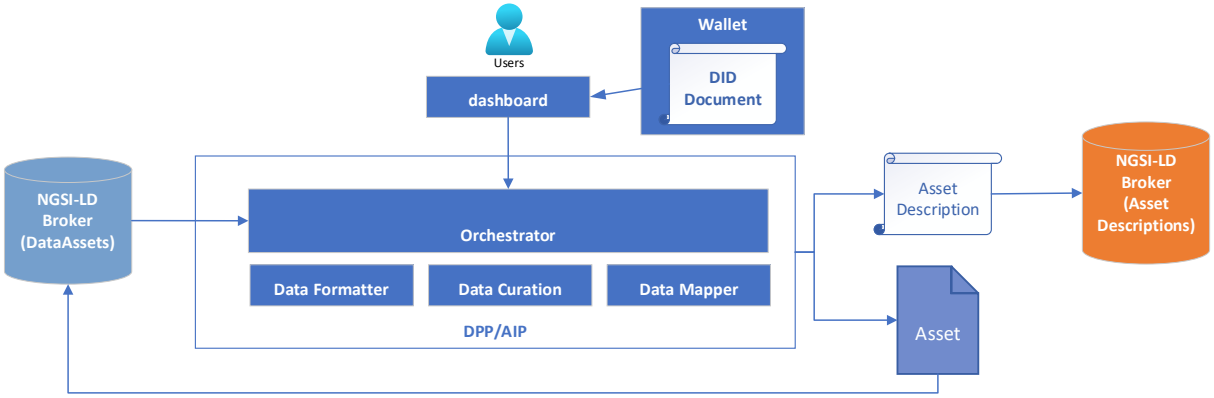


Figure 34 Asset Description Generation

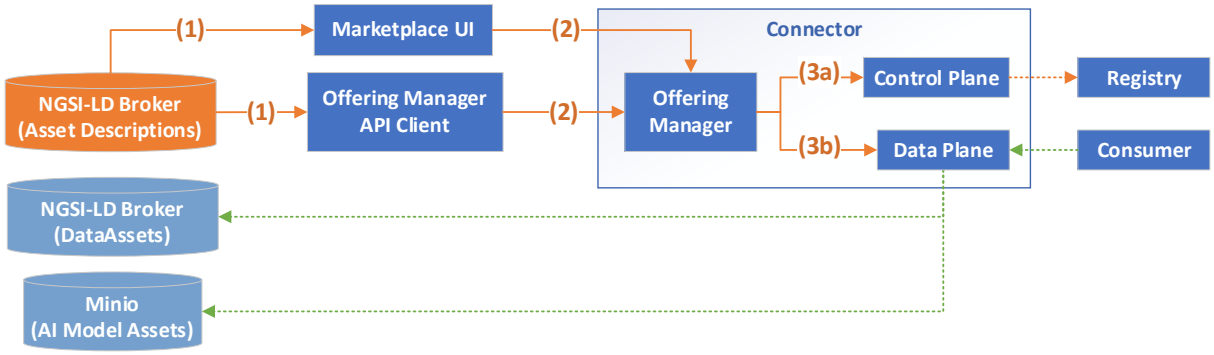


Figure 35 Asset Description Use

The Asset Description holds properties for different aspects of an Asset. Among all types of Assets - i.e. Data, AI Model and Service – common properties are defined.

Figure 36 Common Asset Properties to be captured by Orchestrator

Common properties include aspects of identification, description and tagging to provide context to the Asset. It also holds temporal aspects relating to its creation and modification. Aspect of usage is also provided, as well as provenance relating to dependencies used for the creation of the Asset and what processing has been applied. To support the resolution of the Asset's location within the Provider's storage enablement, endpoint information is also provided.

Modelling/ontology of data assets (information model)

For the DataAsset, specific properties are needed to be captured which relate to temporal aspects, size, periodicity, and whether it is part of another DataAsset.

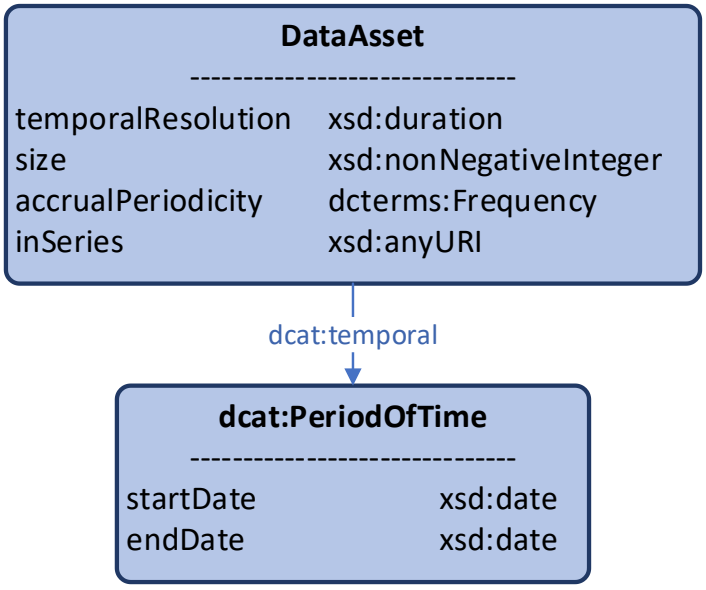


Figure 37 Data Asset Properties

Modelling/ontology of AI assets and training services

Properties that need to be captured for the AI Model Asset are shown in Figure 38. These mainly focus on the intended application of the Asset, specifics about its design such algorithms, input/output parameters, formats it deals with, and whether it handles stream-based DataAssets.

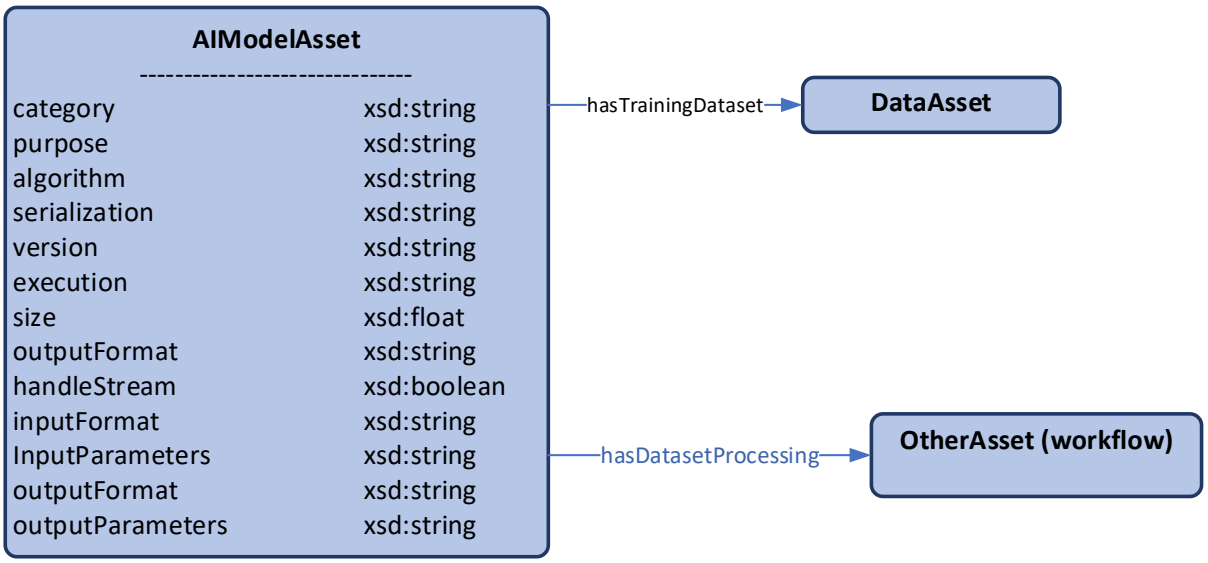


Figure 38 AI Model Asset Properties

As for ServiceAsset properties, interaction details need to be captured, as well as which DataAssets it will provide in return.

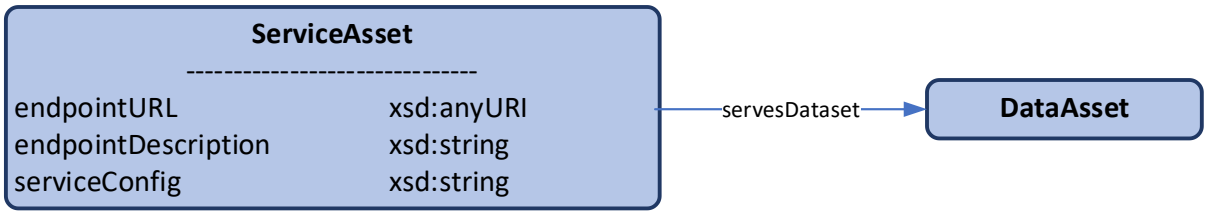


Figure 39 Service Asset Properties

3.2.1.8 Toolbox deployment for NGSI-LD broker

The deployment for the NGSI-LD broker is done through a docker-compose file that deploys all the needed containers for the broker to work.

The broker can be configured through environment variables that are documented in a README file alongside the deployment ones, and are available at the following GitHub repository link: [13]

As introduced in the previous paragraph, the NGSI-LD context broker available as part of the toolbox serves 2 purposes:

- Storing the datasets assets, query-able as batch or stream (through subscription)
- Storing the asset descriptions, to allow the discovery of available datasets through a RESTFUL API based on SEDIMARK Ontology.

Rather than deploying 2 different brokers, it is recommended to make use of tenants such as provided by the Stellio context broker [17]. As defined within the NGSI-LD Specification, the concept of a tenant is that a user or group of users utilises a single instance of an NGSI-LD Context Broker in isolation from other users or groups of users of the same instance, which are considered to be different tenants. Thus, a multi-tenant NGSI-LD system is a system where a single software instance is used by different users or groups of users, the tenants, where any information related to one tenant (e.g. Entities, Subscriptions, Context Source Registrations) are only visible to users of the same tenant, but not to users of a different tenant. Within Stellio, multi-tenancy provides complete isolation of the tenant through the use of different databases across tenants.

The multi-tenant deployment is not integrated by default in the configuration files. It will be evaluated during the initial integration and adjusted if relevant.

3.2.1.9 Toolbox deployment for MageAI

Mage AI will be deployed through a docker-compose configuration, that can be found in the GitHub repository for the SEDIMARK toolbox: [14] The deployment can be configured with the help of the following environment variables

- PROJECT_NAME – Which is the name of the Mage AI deployment, best is to leave it as in the configuration
- REQUIRE_USER_AUTHENTICATION – Specify if the deployment should have authentication or not, the best is to leave it as in the configuration

3.2.1.10 Toolbox deployment for Mage API

Mage API is an API developed inside the SEDIMARK project and is the component that will be used to interact with the Mage AI deployment, in order to automate as much as possible, the interaction with Mage. The Readme for deploying Mage API is available in the following GitHub repository alongside the other toolbox components: [15]

3.2.1.11 Toolbox deployment for MLFlow

As mentioned in Section 3.2.1.7 the deployment of MLFlow will be done through a docker-compose file and the documentation for the configuration and deployment can be found at the following GitHub repository link:

3.2.1.7 the deployment of MLFlow will be done through a docker-compose file and the documentation for the configuration and deployment can be found at the following GitHub repository link: [16]

3.2.1.12 Toolbox deployment for MinIO

As mentioned in Section 3.2.1.1 the deployment of MinIO will be done through a docker-compose file and the documentation for the configuration and deployment can be found at the following GitHub repository link: [21]

An integration with the Context Broker in charge of assets registration will be proposed in order to make assets stored in Minio discoverable. For that purpose, we need a way to match a Minio event with an existing attribute of an existing entity into the Context Broker (to create / update / delete the attribute depending on the event type).

Out of the box, identifiers of a Minio object is the filename which is weak. However, it is possible to add attributes and tags when creating an object. Tags do not seem to be transmitted into notifications, so better use attributes, e.g.:

mc cp --attr "NGSILD-Entity-Id=urn:ngsi-ld:Entity:01;attributeName=MyFlowAsset " MyFlow.zip minio/sedimark

Then the event contains:


```
"object": {
  "key": "MyFlow.zip",
  "size": 2506,
  "eTag": "5e6daae961eae0eb8db73d05ca704d77",
  "contentType": "application/zip",
  "userMetadata": {
    "X-Amz-Meta-Attributename": "MyAsset ",
    "X-Amz-Meta-Ngsild-Entity-Id": "urn:ngsi-ld:Entity:01",
    "content-type": "application/zip"
  },
  "sequencer": "16C70A13B5B27130"
}
```


3.2.1.13 Toolbox deployment for UI Orchestrator

UI Orchestrator is a web-based application designed to simplify the management and creation of AI pipelines by serving as an intuitive wrapper around Mage AI.
Users can tag pipelines created in Mage AI as pre-processing, training, prediction, or streaming, and these tags automatically organize pipelines into corresponding menus in the interface. By accessing these menus, users can render pipelines, execute them by inputting values for variables in each block, and view performance metrics and the execution history. Additional features include pipeline deletion, name editing, and exporting in either CWL or MageAI format.
A key feature of the platform is the ability to create pipelines by linking pre-existing blocks or generating new ones with AI. Users can select blocks from templates or create custom blocks by specifying a type and providing a prompt, with an LLM generating the block for immediate use or saving it as a template. Custom templates can also be added to Mage AI to meet specific needs.
The deployment of the UI Orchestrator is straightforward, ensuring that users can easily integrate it into their existing infrastructure. The application is distributed as a containerized service, enabling quick setup and scalability. To deploy the UI Orchestrator, users need to:

- Prepare the environment: Ensure access to a running Mage AI instance, along with the appropriate credentials and URL.
- Install the application: Obtain the pre-configured Docker image from the SEDIMARK container registry or build the image locally using the provided Dockerfile.

- Configure connection parameters: During the setup, users specify the Mage AI instance details through a simple configuration interface or environment variables, which establish a secure connection between the UI Orchestrator and Mage AI.
- Run the service: Deploy the application using a single Docker command or within a docker-compose setup, as provided in the SEDIMARK documentation.

Once deployed, the UI Orchestrator automatically detects pipelines and integrates seamlessly with Mage AI, offering an intuitive interface for managing workflows. Deployment logs and monitoring tools are included to verify a successful setup and resolve potential issues.

3.2.2 Integration specification

Figures 40-41 illustrate the current progress of tasks required for integrating the AI-related functionalities into the SEDIMARK toolbox. This board will be regularly updated to monitor and manage the integration process effectively.

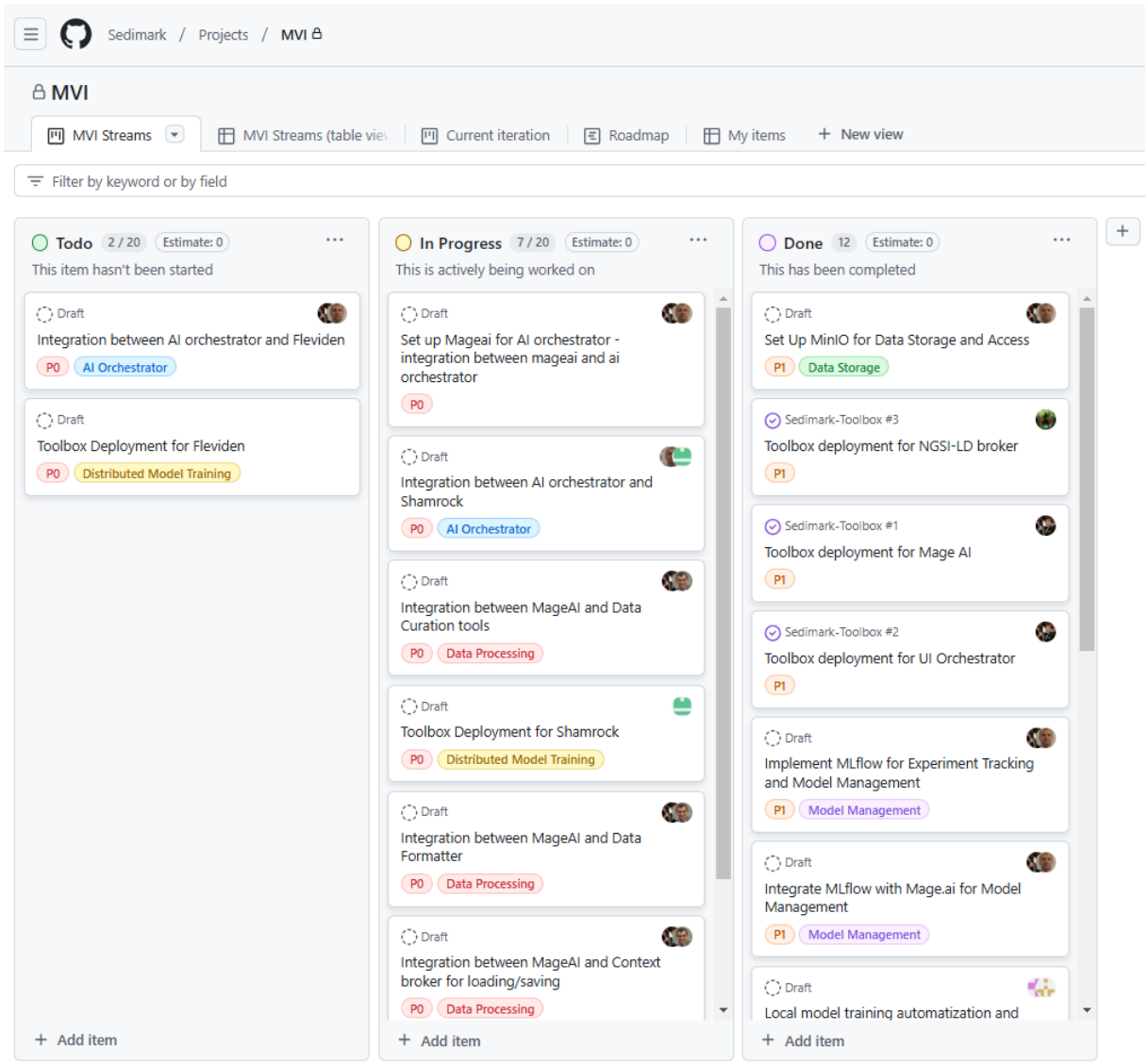


Figure 40 MVI stream task monitoring in GitHub repository (1)

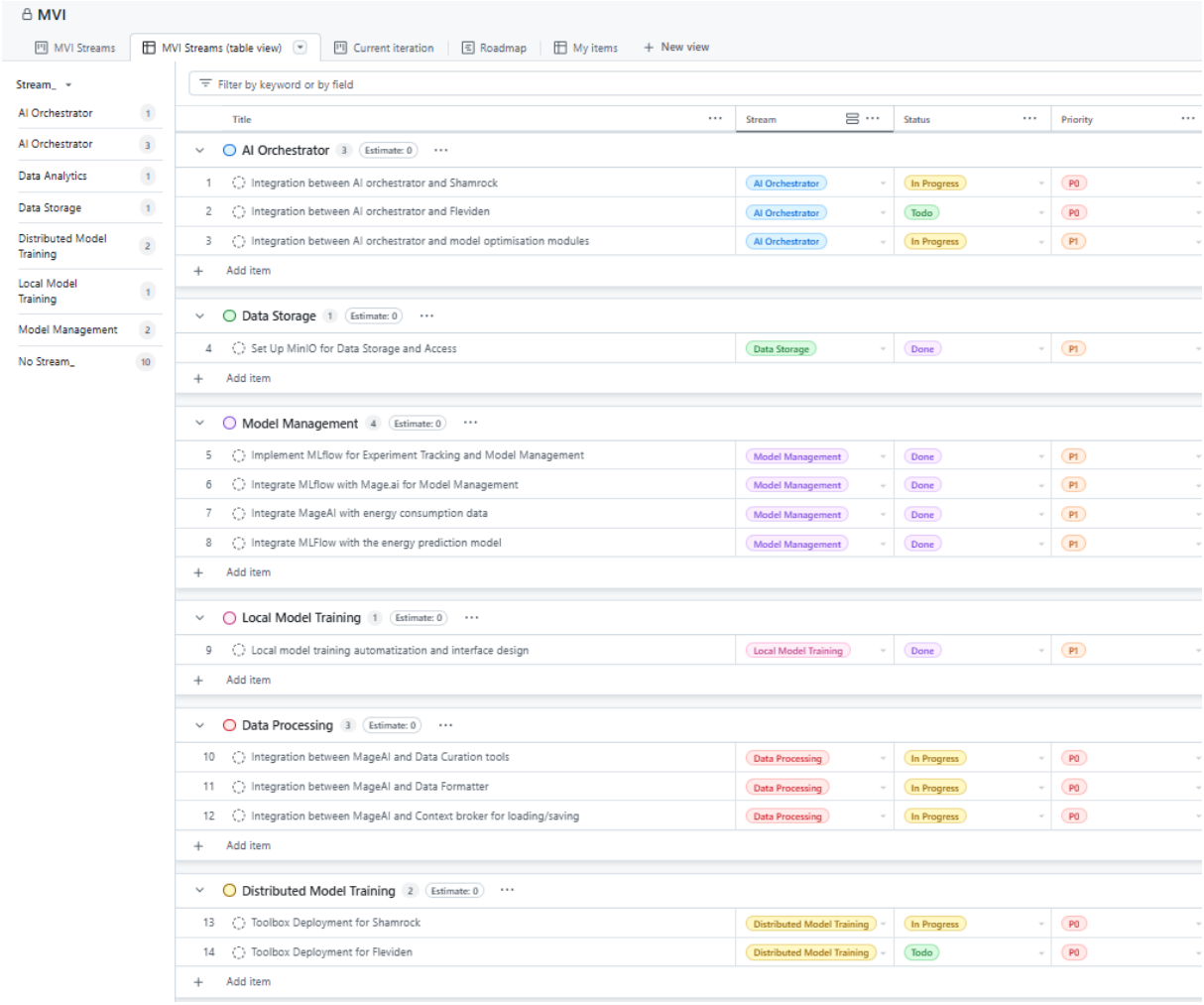


Figure 41 MVI stream task monitoring in GitHub repository (2)

The respective GitHub link is: [17]

3.3 Backend and Frontend Integration Overview

The integration between backend and frontend components plays a crucial role in ensuring a seamless user experience and effective communication between the layers of SEDIMARK marketplace. This section aims to outline the key integration points, the flow of data, and the mechanisms used to establish robust connectivity between the backend systems and frontend interfaces. By detailing the structure and interactions, this section will provide a comprehensive understanding of how the components come together to deliver a cohesive solution.

3.3.1 Define sub-streams and provide a high-level description

This section outlines the various sub-streams that establish the connection between backend components and the SEDIMARK UI as well as the Orchestrator UI, as illustrated in Figure 6. The primary objective of this section is to demonstrate how the backend seamlessly integrates with the frontend to ensure that users of the SEDIMARK marketplace enjoy a smooth and intuitive experience. Additionally, this section will detail the integration between the MVI and the Connector through the SEDIMARK UI, along with the data flow between the connectors.

3.3.1.1 Creation of offering description based on asset description

The creation of an offering description via the marketplace UI can be achieved using its *Publish* section. This section consists of a form, where the user can either select one of the existing asset, she/he created in the NGSI-LD context broker, or create one from scratch. If the latter option is chosen, the marketplace will also take care of pushing the newly created asset description to the NGSI-LD context broker. Then, the user adds policies to rule the access to the data asset, such as a validity period for the contract. Finally, once done filling out this form, the user is invited to review it and confirm her/his will to publish it to the SEDIMARK catalogue.

3.3.1.2 Data flow between data sinks and Connector

As outlined in section 3.1.1.3, interactions between different participant connectors in the dataspace domain are conducted using a peer-to-peer approach. This involves an offering negotiation process via the control plane, followed by an asset exchange process using a data plane which is selected based on the specific asset technology. Our current implementation of the connector data plane supports generic REST APIs as well as MinIO interfaces, which can be targeted by using the MinIO-specific REST API or AWS S3 API. All needed information for the connector to be able to interact with specific asset storage technology is injected into the Offering Manager when registering the offering.

3.3.1.3 Integration between Orchestrator and Context Broker to retrieve asset descriptions

There are two main workflows for handling asset descriptions in the interaction between the Orchestrator UI and the Context Broker: one for creating an asset description and another for retrieving it later if a user opts not to create an offering directly from the description.

Asset descriptions are stored as NGSI-LD entities in the Context Broker, which can be queried to retrieve entities of type "asset description." The Orchestrator UI provides an interface where users can input details about an asset's description, such as whether it pertains to data, AI/ML models, or pipelines. Using this input, the appropriate pipeline is initiated to generate the actual asset. Once the pipeline is completed, the comprehensive asset description is saved to the Context Broker.

If the user chooses to proceed with creating an offering, they will be redirected to the SEDIMARK UI. Otherwise, the Orchestrator UI will display an interface where it queries the Context Broker to retrieve and present all stored asset descriptions in an organised and user-friendly manner. Based on this information, users can then decide to create an offering.

The created assets will be identified locally by their NGSI-LD URI which is a unique identifier of an entity resource with the NGSI-LD Context Broker. The definition of the URI is left to the service creating the asset to define the URI (note: the context broker will enforce the uniqueness of the created URI by refusing creation of a new entity with a URI already in use). There is no naming convention defined at that stage. The need to recommend or enforce a naming convention will be evaluated during the integration.

Once an asset has been created in the Context Broker, it will be accessible through the NGSI-LD interface.

Here is an example of querying the Context Broker URI: [22]

3.3.1.4 Integration between Orchestrator UI and Mage API to create an offering for pipeline assets

In the SEDIMARK toolbox, the representation and actual files of the various pipelines developed within the SEDIMARK platform will be stored directly in Mage AI. This eliminates the need for external storage for pipelines. Mage AI will serve as both a storage solution and a functional component, enabling the creation and execution of pipelines within the toolbox.

To export pipelines from Mage AI, the Mage API will be used to interface with the Connector when a pipeline is downloaded by a user through the Orchestrator UI. In the SEDIMARK platform, there are two main types of pipeline exports: one exports the pipeline in Mage AI format, allowing another user with their own toolbox to directly import the pipeline into their Mage AI; the other exports the pipeline in a standardised format using CWL (Common Workflow Language), enabling users who do not want the full toolbox and are just consumers to utilize the pipelines.

The output of both pipeline export methods will ultimately be a ZIP file containing the pipeline along with all the necessary adjacent objects. However, the process for generating the contents of the ZIP file differs between the two methods. Below is an explanation of the process for each method:

- Mage AI export – The process of exporting directly into Mage AI format is straightforward and it starts by collecting all the files connected to a pipeline that are already available in Mage AI and pack them together in the ZIP file, with a generic README file that explains how to import the pipeline back in Mage.
- CWL export – This method is more complex than the other and begins by converting each block from Mage AI format into standard Python code. Following this, for each block in the pipeline, a corresponding CWL workflow is created. The CWL workflow consists of YAML files representing each block, detailing their inputs, outputs, and a main entry point that defines the environment variables and the sequence for running the blocks. Alongside the CWL pipeline, the ZIP file will include a script to automatically validate and execute the workflow, as well as a README file providing instructions on running the pipeline and visualizing the output.

The process for requesting a pipeline asset begins with the discovery phase, where various connectors holding pipeline assets are identified and presented in the Orchestrator UI. Following this, the consumer initiates the negotiation process and requests the desired pipeline. The Connector in the toolbox where the pipeline resides will query the Context Broker for the pipeline's location, which will call the Mage API. Based on the information provided by the consumer, the pipeline will be exported in the specified format and passed to the consumer's Connector for delivery.

3.3.1.5 Integration between Orchestrator UI, Mage API and Mage AI to create and save data/AI/ML offering (asset) description

The process of saving an asset begins with the user interacting with the Orchestrator UI to create a new data, AI, or ML offering. The user inputs the necessary details, including metadata, associated datasets, and AI/ML model parameters. These inputs are validated by the UI to ensure they meet predefined standards and schemas.

Once validated, the asset description is securely sent to the Mage API, which parses and forwards it to Mage AI. Mage AI processes the input by standardizing formats, enriching metadata through ontologies, and validating datasets or AI/ML models to ensure quality and compliance.

The finalised asset description is stored in the SEDIMARK context broker, with a unique identifier generated by Mage API.

3.3.1.6 Integration between Connector, SEDIMARK UI and AI Orchestrator to retrieve AI/ML assets

In the SEDIMARK ecosystem, AI/ML assets are stored using MinIO for model binaries and the NGSI-LD Broker for metadata and descriptions, enabling efficient and secure access. Trained AI/ML models are saved in MinIO, a high-performance distributed object storage system, where each model is assigned a unique identifier for scalability and easy retrieval. Secure credentials and tokens ensure restricted access to these stored models. Alongside the model binaries, metadata and descriptions are stored in the NGSI-LD Broker. This metadata includes details such as model type, training parameters, performance metrics, versioning, and associated datasets, providing a semantic layer for querying and retrieval.

MLFlow orchestrates the storage process, automating the management of AI/ML lifecycle activities. It saves the trained models to MinIO and simultaneously uploads the metadata and descriptions to the NGSI-LD Broker. Once stored, the metadata is indexed in a central or decentralised catalogue, making the models discoverable through the SEDIMARK Marketplace UI.

The user willing to share his/her AI/ML assets can therefore create an offering in the marketplace UI, in its dedicated *Publish* section. As described in section 3.3.1.1, the user first selects the asset to share from the list of available ones in the NGSI-LD context broker. Then, the user adds policies associated with the offering, i.e. a set of rules to restrict access to the asset, such as a validity period. Upon completion of this form, the user can publish the offering, resulting in the marketplace posting the offering to the catalogue.

Once a particular offering has been discovered, the assets described can be exchanged as explained in section 3.3.1.2.

3.3.1.7 Integration between Orchestrator UI and Shamrock

Running shamrock launches a node process, which connects with other distributed node processes in order to collaboratively train a machine learning model over REST/http. Shamrock follows a standard distributed learning paradigm, where nodes alternate between rounds of training, communication, and aggregation. Shamrock allows for both Federated (where one nodes acts purely as a server/aggregator) and fully decentralised training topologies. On reaching a user specified end condition (e.g. a target accuracy, a number of rounds completed), Shamrock will return both a trained model and any evaluation metrics or meta data contained within the training process. The storage of this model will be handled from the Orchestrator UI which will make a call to the AI Orchestrator in order to save the model and the associated metadata.

From the provider's viewpoint, the user will primarily need to specify a model, a dataset and a choice of distributed learning setup (e.g. Federated Learning). Shamrock accepts as input a) NumPy arrays, b) pandas DataFrame with feature/targets specified as additional parameters, c) csv files that may be loaded into dataframes and d) Pytorch datasets. In the case that raw

data needs additional transformations in order to be ready for modelling, the user will need to provide a set of steps within the Orchestrator UI to transform data into input for that model. Shamrock accepts both Keras and Pytorch models. Further configuration parameters such as choice of learning rate, optimiser etc should also be specified. The user will then start either a Federated or Gossip learning process by launching a Shamrock node within the Orchestrator UI. As the process might likely take some time (a training process might be running for hours to days before a Consumer participates), the process will need to persist beyond the Orchestrator UI. As such, the Orchestrator UI should provide a tab in which to monitor ongoing results from the process. This process will then need to be published to the marketplace as an offering. The key element of the offering will be some form of URL to the address at which the provider is hosting the process, alongside the Shamrock configuration that the provider has used to launch the process. Users wishing to participate in the training process will then need to be able to launch the Orchestrator UI and Shamrock from the offering, launching a node that connects to the provider with the given configuration. The user should have access to a tab where they can monitor the performance of the node that they are running. As Shamrock is capable of sharing full model definitions, it is not strictly necessary that the model itself is included in the offering. However, if steps need to be taken in order to transform input data for the model, these should be included. Otherwise, a Consumer might have to engage in a process of trial and error to find transformations of their data that work for the model. These steps can be captured as a Mage pipeline and shared as an asset, or perhaps as part of the offering description.

3.3.1.8 Integration between Orchestrator UI and Fleviden

The process of starting the federated training process will begin from the Orchestrator UI. Once the server entity is defined, it will wait for a minimum of end-users / clients to be subscribed. When this condition is reached, the server will share an initial model from the supported frameworks with the clients, which will be in charge of training this model over a set number of epochs. Once they complete their training, they will send the updated weights to the server, which will aggregate them. This will be repeated during a set number of iterations or rounds. During this process, new clients can join the network, as the server will continue waiting for requests. The framework also supports client unsubscription in the middle of the process. The server will monitor the health of the different nodes periodically. In each round, not all clients have to compulsory contribute to the training process. The active clients in each round will depend on their availability, training time, etc.

Once the number of training rounds has been reached, the final aggregated model will be sent back to all the clients who participate in the process, and it will be able to be processed using 3rd party apps (MinIO, MLFlow, etc.), as currently the tool gives support to them. The storage of this model will be handled from the Orchestrator UI which will make a call to the AI Orchestrator in order to save the model and the associated metadata.

3.3.1.9 Integration between Recommender and SEDIMARK UI

SEDIMARK currently provides a content-based recommender system. The recommender system in SEDIMARK serves two recommendation purposes: (i) finding assets based on the queries provided by the user; and (ii) recommending similar assets based on the given asset. The SEDIMARK recommender system module currently implements several algorithms for term-based information retrieval. These include the following:

- Latent Semantic Indexing - a method based on natural language processing. The goal of this approach is to analyse the relationships between the assets based on their descriptions and the terms those descriptions contain.
- Sentence Transformers Models - machine learning models generating dense vector embeddings to capture the semantic meaning of sentences using numerical representations. SEDIMARK's recommender systems module implements three-sentence transformer models.

To offer a SEDIMARK user a tailored option, the user has the choice to use any of these recommendation algorithms.

The output of the recommendation module is a list of suggested assets that are the best matches to either the user-defined query or the provided asset.

The communication between GUI and the Recommender system will be done through the Redis queue. The GUI will put any user actions, such as user searches or queries on a Redis queue. Additionally, the Redis queue will be periodically updated with changes to the local asset catalogue. The output of the recommender system will be added to the Redis queue in a JSON format that the GUI will subsequently process and display to the user.

3.3.1.10 Integration between SEDIMARK UI and Connector

The SEDIMARK UI interacts with the data space connector to provide users with convenient interfaces to manage their offerings. Any offering where a given user is involved, either as a provider or as a consumer, appears on her/his *dashboard*. This dashboard consists of several sections, all accessible via a sidebar, where users can:

- See an overview of their offerings.
- Review their offerings provided.
- Manage their consumed offerings, and more specifically request a transfer of their data-to-data sinks of their choice.

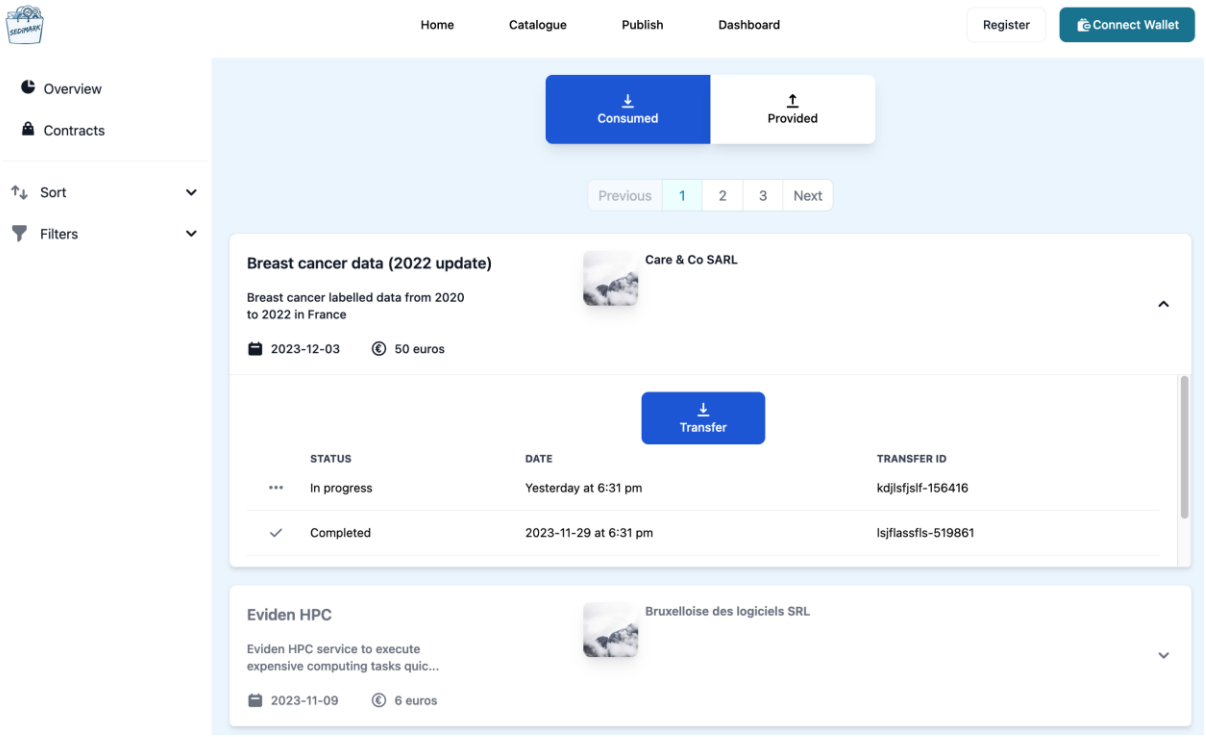


Figure 42 Example of marketplace dashboard UI to manage consumed offerings

The SEDIMARK marketplace code and deployment manifests can be accessed in its repository: [5]

3.3.2 Integration specification

Figures 43-44 present the current status of the tasks that need to be completed in order to integrate the frontend of the SEDIMARK Marketplace with the SEDIMARK toolbox. This board will be continuously updated to keep track of the integration progress.

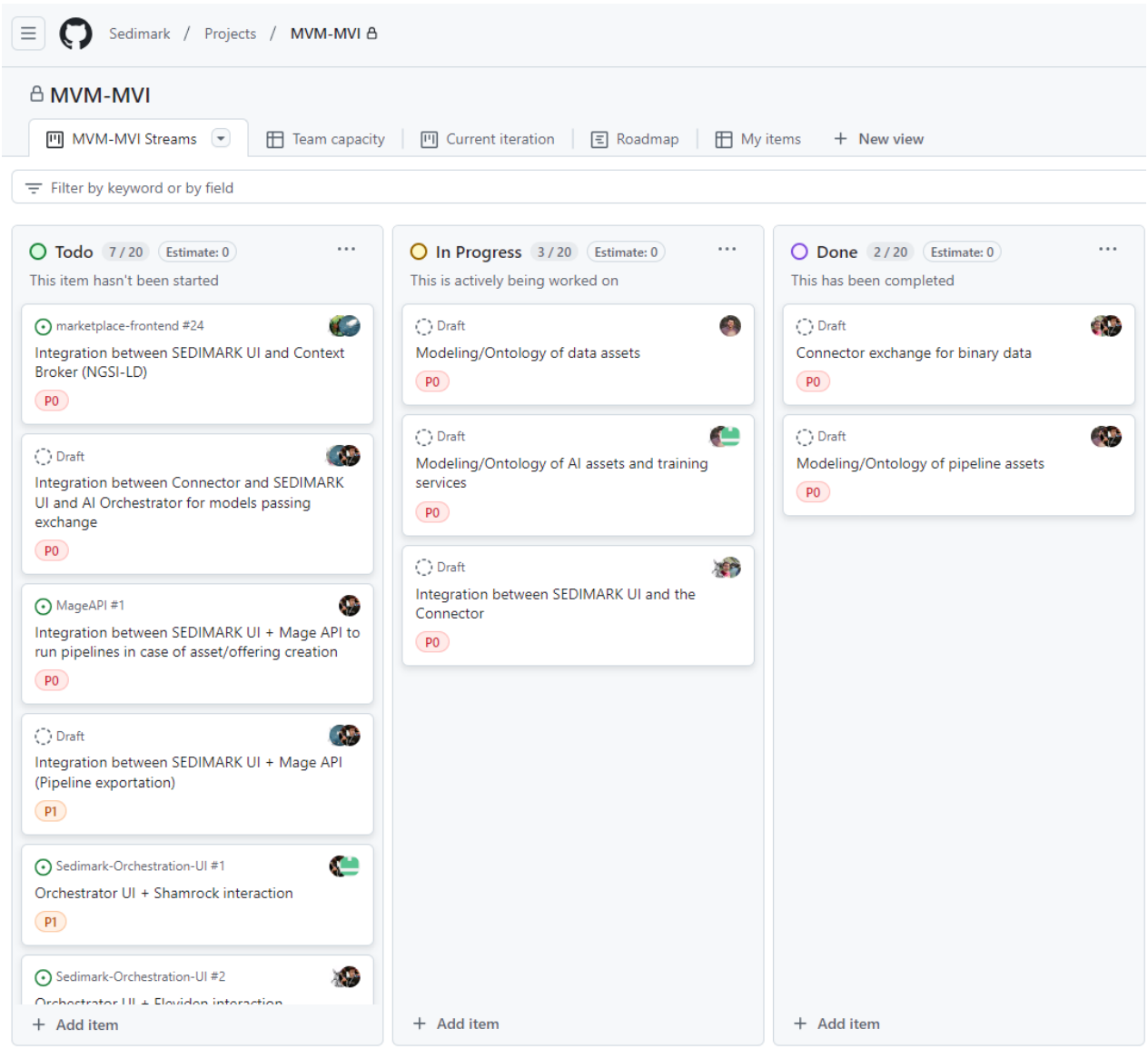


Figure 43 The MVM-MVI issue dashboard on the git repository (1)

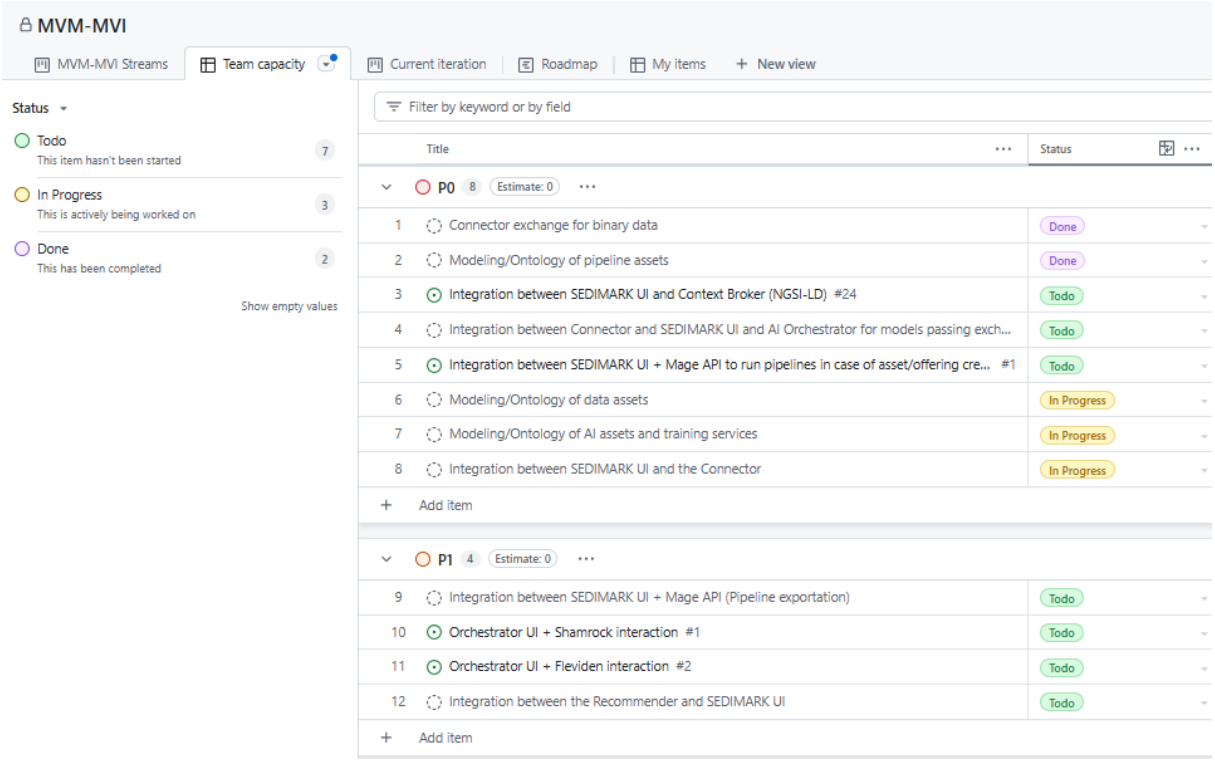


Figure 44 The MVM-MVI issue dashboard on the git repository (2)

The relevant link for the GitHub repository is: [19]

4 Cross-check the updated functional and non-functional requirements

The scope of this section is to provide and update the functional and non-functional requirements of SEDIMARK architecture related to the first integrated release. The requirements will be mapped per stream in order to acquire a common view of which stream each requirement interacts with. For this purpose, two tables are provided (Table 1, Table 2) correlating all the high-priority requirements (H-REQ) and medium-priority requirements (M-REC) that were promised to be fulfilled in the second version, with the three streams.

Table 1 Functional requirements status of fulfilment

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-SEC-01	Authentication of users	H-REQ	• Identity management	MVM	Fully
Req-SEC-02	Authorisation policies of assets	H-REQ	• Trust management	MVM	Partially
Req-SEC-03	Origin of assets	H-REQ	• Trust management • IOTA Client	MVM	Fully
Req-SEC-04	Trust Metadata on Distributed Ledger	H-REQ	• Trust management • Data integrity • Registry • IOTA Client	MVM	Fully
Req-SEC-05	Decentralised Provisioning	H-REQ	• Contracting • Smart Contracts • Service Request • Service Provisioning • IOTA Client • Transactions	MVM	Partially
Req-SEC-06	Secure channel of the assets	H-REQ	• Data encryption • Data integrity • Offering Sharing	MVM	Fully
Req-DP-01	Data cleaning tools	H-REQ	• Data curation • Data profiling • Data processing dashboard	MVI	Fully

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-DP-02	Flexibility to handling both static and streaming data	M-REC	• Data curation • Data orchestrator • Data profiling • Data processing dashboard	MVI	Partially
Req-DP-03	Configurable data processing pipeline	M-REC	• Data curation • Data processing dashboard • Data orchestrator • Data profiling • Data adapter	MVI	Fully
Req-DP-04	Data quality indicators	M-REC	• Data processing dashboard • Data quality evaluation • Data integrity • Annotation • Data profiling • Data augmentation	MVI	Partially
Req-DP-05	Adaptability of data cleaning mechanisms	M-REC	• Data quality evaluation • Energy efficiency • Data augmentation • Data profiling • Data orchestrator	MVI	Fully
Req-DP-07	Data cleaning modules extendable definitions	M-REC	• Data curation • Data processing dashboard • Data adapter • Data quality evaluation • AI orchestrator	MVI	Fully
Req-DP-09	Dataset augmentation	M-REC	• Data orchestrator • Data processing dashboard	MVI	Fully

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-DP-10	Anonymisation of private information	H-REQ	• Data profiling • Data anonymisation • Data encryption • Data processing dashboard • Data orchestrator	MVI	Partially
Req-DP-11	External Metadata for curation	M-REC	• Data curation • Data processing dashboard • Annotation	MVI	Fully
Req-DP-12	Load and save data from storage	H-REQ	• Data loader • Data saver	MVI	Fully
Req-ML-01	Model input data cleaning and formatting	H-REQ	• Data curation • Data profiling • Annotation • Data adapter • Feature engineering • Semantic enrichment • Data augmentation • Local model training • Distributed model training	MVI	Fully
Req-ML-02	Decentralised ML	M-REC	• Distributed model training • Formatting • Model inference • AI as a service	MVI	fully
Req-ML-03	Trusted participation in decentralised training	H-REQ	• Distributed model training • Trust management • Identity management • Data encryption • AI as a service	MVI-MVM	Partially

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-ML-04	Models agnostic to platforms	M-REC	• Model optimisation • Local model training • Distributed model training • Frugal AI • Formatting • AI orchestrator	MVI	Partially
Req-ML-05	Models persistence mechanisms	H-REQ	• Distributed model training • Local model training, • Data storage • Formatting • Data analytics • AI as a service	MVI	Partially
Req-ML-06	Event generation from pattern extraction	H-REQ	• Data analytics • Semantic enrichment • Model optimisation • Model inference	MVI	Partially
Req-ML-08	Variable Distributed learning scenarios (federated and gossip)	M-REC	• Distributed model training • AI orchestrator • AI as a service • Trust management • Peer discovery	MVI	Partially
Req-ML-09	Multiple initialisation options for distributed training	M-REC	• Distributed model training • AI orchestrator • AI as a service • Offering description • Offering discovery • Offering sharing	MVI	Partially
Req-ML-12	Model evaluation and validation	M-REC	• Model optimisation • Model inference • Data analytics • AI model validation	MVI	Partially

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-RS-01	User profiling	H-REQ	• User profiling • Logging • Recommendations	MVM-MVI	Partially
Req-RS-02	Rich item information	H-REQ	• Recommendations • Offering discovery • Offering statistics • Monitoring • Ratings	MVM	Partially
Req-RS-03	Decentralised Recommender system	H-REQ	• Distributed model training • Recommendations • Data encryption • Data anonymisation • Trust management	MVM-MVI	fully
Req-RS-04	Cold start problem	H-REQ	• Recommendations • Distributed model training • AI model optimisation	MVM-MVI	Partially
Req-RS-05	Similarity of assets	M-REC	• Recommendations • Offering discovery	MVM-MVI	Partially
Req-RS-06	Recommend different types of assets	M-REC	• Recommendations • Offering discovery • Offering statistics	MVM-MVI	Partially
Req-RS-07	Employ multiple recommendation models based on user actions	M-REC	• Recommendations • Offering discovery • Offering statistics • User profiling • Logging	MVM-MVI	Partially
Req-RS-08	Personalised recommendation based on user feedback	M-REC	• Recommendations • User profiling • Logging • Data analytics	MVM-MVI	Partially

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-RS-09	Dynamic update of recommendations	M-REC	• Recommendations • Real time data processing • Distributed model training • Data processing dashboard	MVM-MVI	Partially
Req-RS-10	Cross-domain recommendation	M-REC	• Recommendations • Offering discovery • Data analytics • Semantic enrichment	MVM-MVI	Partially
Req-EE-01	Lightweight and energy efficient DP modules	H-REQ	• Data curation • Data augmentation • Data profiling • Data orchestrator • Data processing dashboard • Model optimisation • AI orchestrator	MVI	Fully
Req-EE-02	Lightweight and energy efficient AI/ML models	M-REC	• Local model training • Distributed model training • Frugal AI • Model inference • Model optimisation	MVI	Partially
Req-EE-03	Energy efficient decentralised training of ML model	H-REQ	• Model optimisation • Model inference • AI orchestrator	MVI	Partially
Req-EE-06	Energy efficient real time data processing	M-REC	• Data processing orchestration • Data processing dashboard • Frugal AI • Energy efficiency	MVI	Partially

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-INT-01	Information model for data and their metadata	H-REQ	• Data quality evaluation • Formatting • Semantic enrichment • Annotation • Data processing dashboard • Data orchestrator • Offering validation	MVI	Fully
Req-INT-02	Metadata fields	H-REQ	• Data quality evaluation • Semantic enrichment • Annotation • Data anonymisation • Offering validation	MVI	Fully
Req-INT-03	Data compliance with the information model	H-REQ	• Data quality evaluation • Formatting • Semantic enrichment • Annotation • Data anonymisation • Offering validation	MVI MVM	Partially Partially
Req-INT-04	Enforcing data compliance with the information model	M-REC	• Annotation • Formatting • Offering validation • Data mapper • Data processing dashboard • Data orchestrator • Data quality evaluation	MVI MVM	Fully

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-INT-05	Information model for AI models	H-REQ	• Distributed model training • Formatting • Annotation • Model optimisation • Model inference • Offering validation	MVI	Partially
Req-STR-01	Default dataset storage domain	H-REQ	• Data storage • Distributed storage	MVI	Partially
Req-STR-02	Storage of offering descriptions on distributed catalogue	H-REQ	• Data storage • Offering description • Catalogue • Offering registration • Offering discovery	MVM	Fully
Req-STR-03	Temporary storage of intermediate artefacts with pipeline	H-REQ	• Data storage • Feature engineering • Data orchestrator	MVI	Partially
Req-STR-04	Storage of post-processed data in consumable manner	H-REQ	• Data storage • Model inference • Formatting • Validation	MVI	Partially
Req-STR-06	Storage Service for constrained data providers	M-REC	• Distributed storage • Service provisioning • Offering sharing	MVI	Partially
Req-STR-07	Storage for knowledge domain services	M-REC	• Data storage • Semantic enrichment • Catalogue	MVI	Partially

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-STR-08	Storage for offerings other than datasets	M-REC	• Data storage • Feature engineering • Local model training • Model inference • Formatting • Model optimisation • Validation	MVI	Partially
Req-STR-09	Storage of artefacts for enabling security and trust	H-REQ	• Data Storage • DLT • Trust	MVM	Fully
Req-STR-10	Storage of artefacts for Marketplace management	H-REQ	• Data Storage • Marketplace	MVM	Fully
Req-P&D-01	Assets described as part of offerings	H-REQ	• Offering description • Offering discovery	MVI-MVM	Partially
Req-P&D-02	Offerings' registry	H-REQ	• Offering registration • Offering sharing • Offering discovery • Registry • Distributed storage • Catalogue • Data storage	MVM	Fully
Req-P&D-03	Generic offering metadata	H-REQ	• Offering description	MVI-MVM	Partially
Req-P&D-04	Open Data portal discovery	H-REQ	• Open data enabler	MVI-MVM	Partially

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-P&D-05	Offerings' catalogue for queries	H-REQ	• Offering discovery • Catalogue	MVM	Fully
Req-UI-01	Logging in UI	H-REQ	• Identity management • Frontend	MVM	Fully
Req-UI-02	Offerings discoverability	H-REQ	• Offering discovery • Frontend	MVM	Fully
Req-UI-03	Users' identity management	H-REQ	• Identity management • Frontend	MVM	Fully
Req-UI-04	Offerings management	H-REQ	• Offering registration • Trust management • Frontend • Payment/Billing	MVM	Fully
Req-UI-05	Offering description page	H-REQ	• Offering discovery • Data visualisation • Frontend	MVM	Partially
Req-UI-06	SEDIMARK toolbox access in UI	M-REC	• Offering discovery • Data visualisation • Frontend	MVM	Fully
Req-SCT-01	Smart Contracts support	M-REC	• Smart contracts • Transactions	MVM	Fully
Req-SCT-02	Tokenisation of Assets	M-REC	• Smart contracts • Tokenisation	MVM	Fully
Req-SCT-03	User Digital Wallet	M-REC	• Tokenisation • Payment	MVM	Partially

Table 2 Non-functional requirements status of fulfilment

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-NF-01	Decentralisation	H-REQ	• All DLT layer modules • Distributed model training • Distributed storage • Catalogue	MVM	Partially
Req-NF-02	Security, Privacy, Trust	H-REQ	• All security/trust layer modules	MVM	Fully
Req-NF-03	Interoperability	H-REQ	• Annotation • Data adapter • Semantic enrichment • Validation • Formatting	MVI	Partially
Req-NF-04	Data availability and quality	H-REQ	All data processing modules	MVI	Fully
Req-NF-05	Intelligence	H-REQ	All AI layer modules	MVI	Fully
Req-NF-06	Energy efficiency	H-REQ	• Energy efficiency (module in data processing) • Frugal AI	MVI	Partially
Req-NF-07	Resilience and Reliability	H-REQ	N/A (mapped to all components and the platform as a whole)	MVM MVI	Fully at the current integration phase
Req-NF-08	Scalability	H-REQ	N/A (mapped to all components and the platform as a whole)	MVM MVI	Fully at the current integration phase
Req-NF-09	Openness, Extensibility	H-REQ	N/A (mapped to the platform as a whole)	MVM MVI	Fully at the current integration phase

Identifier	Short Name	Priority-Req. Level	Functional components	Related stream(s)	How the stream(s) fulfil(s) the requirement (fully, partially, at all)
Req-NF-10	Usability	H-REQ	All marketplace service modules	MVM MVI	Fully at the current integration phase
Req-NF-11	Maintainability	H-REQ	N/A (mapped to all components and the platform as a whole)	MVM MVI	Fully at the current integration phase
Req-NF-12	Adaptivity to data types and fast processing	M-REC	All data processing modules	MVM MVI	Fully at the current integration phase
Req-NF-13	Reusability	H-REQ	All asset sharing and discovery modules	MVM MVI	Fully at the current integration phase
Req-NF-14	Flexibility	H-REQ	N/A (mapped to all components and the platform as a whole)	MVM MVI	Fully at the current integration phase

5 Integration plan for the final release

The SEDIMARK platform has been carefully designed with the principles of microservice architecture to enable seamless deployment and efficient scaling. Each software component within the platform is encapsulated in its own environment, facilitating modularity and adaptability to meet diverse user needs. To support this approach, each component is associated with a dedicated GitHub repository. These repositories contain a Docker file that allows users to build and run a local container of the respective component. Additionally, comprehensive documentation is provided to guide users through the steps required to construct, configure, and deploy the container images effectively. To further streamline the deployment process, pre-built Docker images are hosted in the GitHub container registry under the SEDIMARK organisation. This eliminates the need for users to manually construct images, significantly reducing setup time and complexity.

For components that rely on external software dependencies, such as MinIO or PostgreSQL, a docker-compose file is included to allow users to quickly deploy a fully operational instance. This ensures that all necessary services and dependencies are configured and ready for use without requiring extensive manual effort. By automating these steps, the platform guarantees a consistent and reproducible deployment process across all use cases and environments.

Given the large number of components that make up the SEDIMARK platform, an iterative deployment strategy has been adopted. This approach begins with the containerisation of individual components, ensuring that each one can function independently while maintaining compatibility with the overall architecture. Once this step is complete, manifests are created for deploying and configuring the components required for the platform's two primary streams: the Minimum Viable Intelligence (MVI) and the Minimum Viable Marketplace (MVM). These streams are managed independently during the development and deployment phases to streamline the integration process and ensure modular functionality. Each stream has a dedicated repository containing its respective docker-compose file, along with all necessary resources and documentation to configure, execute, and test its components.

The final deployment phase integrates the manifests for both the MVM and MVI streams to achieve a fully operational version of the SEDIMARK platform. This integration consolidates the functionalities of both streams, creating a cohesive system capable of supporting the platform's diverse use cases and meeting its scalability requirements. This iterative and modular approach ensures that the deployment process remains manageable and efficient while maintaining flexibility for future enhancements and expansions. By adopting this strategy, the SEDIMARK platform achieves a robust deployment framework that ensures reproducibility, scalability, and ease of use. These principles are crucial to supporting the platform's pilots and broader adoption across diverse sectors, making the final release a technically sound and user-friendly system capable of addressing complex challenges in a decentralized data and services marketplace.

6 Conclusions

This deliverable has outlined the progress made in the second integrated release of the SEDIMARK platform, building on the foundational work of earlier phases and demonstrating significant advancements across its key components. By transitioning from independent Proof-of-Concept scenarios to structured streams; Minimum Viable Marketplace (MVM), Minimum Viable Intelligence (MVI), and the combined MVM-MVI analysing the backend and frontend integration, SEDIMARK has achieved greater cohesion, scalability, and functionality in its decentralised data and services marketplace.

The MVM stream has ensured robust marketplace functionalities such as participant onboarding, offering registration, and secure asset exchanges, all of which are underpinned by Distributed Ledger Technology (DLT) for trust and transparency. Meanwhile, the MVI stream has brought intelligence to the platform through capabilities like local and distributed model training, federated learning, and model performance, emphasizing privacy-preserving AI approaches. The integration of these streams into the combined MVM-MVI ecosystem demonstrates SEDIMARK's capability to support complex use cases and deliver actionable insights to participants.

Furthermore, this deliverable highlights the alignment of platform development with functional and non-functional requirements, showcasing the successful implementation of high-priority features while ensuring adaptability to evolving project needs. The continuous integration and testing efforts have been validated through practical pilot applications, ensuring readiness for deployment in real-world scenarios.

Looking ahead, the insights and feedback from this release will inform the final integration phase, focusing on enhanced modularity, seamless user interaction, and performance optimisation. Through these iterative developments, SEDIMARK is positioned as a robust, secure, and intelligent marketplace, contributing to the realisation of the European Union's vision for a decentralised and interoperable data economy.

7 Bibliography

[1] SEDIMARK, Deliverable 5.2: Integrated releases of the SEDIMARK platform. First version, April 2024.

[2] SEDIMARK, Deliverable 2.3: SEDIMARK architecture and interfaces. Final version, September 2024.

[3] SEDIMARK, Deliverable 5.1: Evaluation methodology, metrics and integration plan, November 2023.

[4] GitHub repository for smart contracts, [Online]. Available: https://github.com/Sedimark/sedimark-smart-contracts

[5] GitHub repository for marketplace frontend, [Online]. Available: https://github.com/Sedimark/marketplace-frontend

[6] GitHub repository for DLT Booth, [Online]. Available: https://github.com/Sedimark/dlt-booth

[7] GitHub repository for Registry, [Online]. Available: https://github.com/Sedimark/hornet-extra

[8] GitHub repository for Issuer, [Online]. Available: https://github.com/Cybersecurity-LINKS/mediterraneus-issuer

[9] GitHub repository for Offering Manager, [Online]. Available: https://github.com/Sedimark/offering-manager

[10] GitHub repository for Offering Catalogue, [Online]. Available : https://github.com/Sedimark/catalogue-coordinator

[11] GitHub repository for the Catalogue, [Online]. Available: https://github.com/Sedimark/catalogue

[12] GitHub repository for the EDC connector, [Online]. Available: https://github.com/Sedimark/sed-edc-connector

[13] GitHub repository for the NGSI-LD broker, [Online]. Available: https://github.com/Sedimark/Sedimark-Toolbox/tree/main/ngsild_broker_deployment

[14] GitHub repository for Mage.AI, [Online]. Available: https://github.com/Sedimark/Sedimark-Toolbox/blob/main/dp_ai_pipeline_orchestration_deployment/docker-compose.yaml

[15] GitHub repository for Mage.AI (readme), [Online]. Available: https://github.com/Sedimark/Sedimark-Toolbox/blob/main/dp_ai_pipeline_orchestration_deployment/README.md

[16] GitHub repository for MLFlow, [Online]. Available: https://github.com/Sedimark/Sedimark-Toolbox/blob/main/dp_ai_pipeline_orchestration_deployment/README.md

[17] Multitenancy in STELLIO Context Broker https://stellio.readthedocs.io/en/latest/user/multitenancy.html

[18] GitHub repository for MVM stream https://github.com/orgs/Sedimark/projects/3

[19] GitHub repository for MVI stream https://github.com/orgs/Sedimark/projects/5/views/2

[19] GitHub repository for MVM-MVI stream https://github.com/orgs/Sedimark/projects/6

[20] GitHub repository for local model training https://github.com/Sedimark/local_model_training

[21] GitHub repository for MinIO https://github.com/Sedimark/Sedimark-Toolbox/blob/main/dp_ai_pipeline_orchestration_deployment/README.md

[22] Context Broker URI http://localhost:8080/ngsi-ld/v1/types/asset