D2.4 - Interoperable Data Spaces Framework

WP 2 – Open-source Interoperability Toolkit

T 2.4 – Interoperable Data Spaces Framework

Submission date: 31.03.2024

Project Acronym	INTERSTORE
Call	HORIZON-CL5-2022-D3-01
Grant Agreement N°	101096511
Project Start Date	01-01-2023
Project End Date	31-12-2025
Duration	36 months

INFORMATION

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

Written By	Marcantonio La Franca (ENG) Ferdinando Bosco (ENG) Giovanni Di Marco (ENG)	2024-02-05 2024-02-05 2024-03-07
Checked by	Matjaz Juric (SUN)	2024-03-20
Reviewed by	Matjaz Juric (SUN) Francesco Guaraldi (ENX)	2024-03-21 2024-03-24
Approved by	Antonello Monti (RWTH) – Project Coordinator	2024-03-27
Status	Final Version 1.0	2024-03-26

DISSEMINATION LEVEL

CO	Confidential	
CL	Classified	
PU	Public	X

VERSIONS

Date	Version	Author	Comment
05-02-2024	0.1	Marcantonio La Franca (ENG) / Ferdinando Bosco (ENG)	The first draft with TOC
19-02-2024	0.2	Marcantonio La Franca (ENG) / Ferdinando Bosco (ENG)	Add draft content to every chapter
28-02-2024	0.3	Marcantonio La Franca (ENG) / Ferdinando Bosco (ENG)	Add Platform architecture information and User Interface Screen
07-03-2024	0.4	Marcantonio La Franca (ENG) / Ferdinando Bosco (ENG) / Giovanni Di Marco (ENG)	Add detail of Platform Architecture, Data Models, APIs, Language & Framework

08-03-2024	0.5	Marcantonio La Franca (ENG) / Ferdinando Bosco (ENG) / Giovanni Di Marco (ENG)	Integrate information
21-03-2024	0.6	Matjaz Juric (SUN)	Review & Feedback
24-03-2024	0.7	Francesco Guaraldi (ENX)	Review & Feedback
26-03-2024	1.0	Marcantonio La Franca (ENG)	Revisions Accepted

ACKNOWLEDGEMENT


InterSTORE is a EU-funded project that has received funding from the European Union's Horizon Research and Innovation Programme under Grant Agreement N. 101096511.

DISCLAIMER

ABBREVIATIONS AND ACRONYMS

WP	Work Package
IDS	International Data Space
TrueConnector	TRUsted Engineering Connector
API	Application Programming Interface
RAM	Reference Architecture Model

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The digitalization of the energy sector demands higher level of operational excellence with the adoption of disruptive technologies to foster cross-domain data sharing and data driven innovation.
Following key elements in data management in support of a data economy need to be fulfilled:

- Data models / Semantics: Defining an appropriate data model beyond a single sector is a key ingredient for interoperability.
- Context Information: Defining the context is a key ingredient for bringing the gap between different verticals.
- Data Sovereignty: The ability of a data owner to define what a third party is allowed to do with her/his data.
- Open API: Closed solutions will not create a real open and competitive market. Open APIs offer the perfect bridge between private infrastructure spaces.

Under the Technological perspective its crucial to define standards regarding, protocols, interoperability, and integration as well as to define specific and transparent rules for data ownership, data security and data exchange.

In this context, the Interoperable Data Space Framework paves the basis for implementing a secure and trusted data space ecosystem, in which energy stakeholder can participate, cooperate and collaborate, exchanging data and implementing new business services.

The Interoperable Data Space Framework consists of two main components: the Data Space Middleware and the Energy Data Space Connector.

It leverages on the IDS Reference Architectural Model as well on the FIWARE NGSI Standardized interfaces for implementing a seamless end-to-end data exchange, ensuring to data owner a full control and management over its own data.

The first version of the Energy Data Space Connector described in this document, includes several components and functionalities, among these:

- IDS Components for IDS compliance and end-to-end connections
- FIWARE components for NGSI standardization and data exchange
- Open APIs for the integration of the Data Space Middleware services, as well for the integration of external platforms.
- Data Services module includes several services for supporting the data exchange within the Energy Data Space Connector, such as data harmonization and blockchain-based notarization services.
- Graphical User Interface, supporting the configuration of the data space environment, the discovery of data sources and data-driven services, as well as the capabilities for publishing data resources in a standard and well recognised way.

This first version of the Interoperable Data Space Framework will be tested and evaluated within several use cases within the Interstore pilots and a new version, which will implement additional requirements, also considering the feedback of the first validation phase, will be provided in March 2025.

1 Introduction

This document describes the Interoperable Data Space Framework and serves as a comprehensive guide to the architectural and design aspects of the software, outlining the key decisions, structures, and methodologies employed in its development. It also describes how to configure and launch the Energy Data Space Connector in a test environment.

1.1 Purpose and Scope of the document

The primary purpose of this document is to provide a detailed blueprint of the software's design, enabling all relevant parties to gain a deep understanding of the system's architecture. It acts as a reference for the project partners and future interested teams, ensuring consistency in implementation and facilitating future maintenance and enhancements.

This document covers the design considerations for Energy Dataspace Connector, a connector specifically designed and implemented for the energy sector. It outlines the high-level architecture, major components, data flow, and interactions between different modules to provide a holistic view of the software.

Data ecosystems follows a data-driven approach. The purpose of Energy Dataspace Connector is to enable collaboration and cooperation among diverse, interconnected participants (stakeholders) that depend on each other for their mutual benefit, supporting trusted and secure data sharing.

1.1.1 Energy Sector and The Digitalization Process

- Integration of Renewable Energy Sources: facilitates the coordination of renewable energy generation, storage, and consumption.
- Demand Response and Energy Efficiency: enables energy providers to collect and analyze information about energy consumption patterns, demand fluctuations, and customer behaviors.
- Grid Management and Operation: Data exchange plays a pivotal role in grid management and operation. Real-time data on power generation, transmission, and distribution is shared among various stakeholders.
- Market Operations: data exchange is essential for efficient market operations and trading. Market participants need access to accurate and timely data on energy prices, demand forecasts, and market conditions.
- Asset Management: stakeholders can collaboratively plan and optimize the expansion, upgrade, and maintenance of energy assets.

The Energy Data Space connector is an ecosystem in which stakeholders can share and access data while ensuring data protection, privacy, and compliance with relevant regulations.

It encompasses the following building blocks to fulfil its functionality:

- Data Storage and Access Control
- Data Exchange and Integration
- Data Privacy and Security
- Interoperability and Standards

1.2 Task 2.4

The objective of this task is to establish a structure that facilitates seamless data interoperability among various participants involved in InterSTORE processes. The envisioned approach involves utilizing data spaces and capitalizing on outcomes generated by tasks in other work packages (WPs). The framework will draw upon existing open-source software components, notably FIWARE-based technology enablers created by ENG, with the primary element being the TrueConnector (TRUsted Engineering Connector) for the International Data Space (IDS) ecosystem.

The TrueConnector facilitates data exchange within the IDS ecosystem through an execution core container and a data application back-end module. This setup is extended and improved for enabling InterSTORE applications to generate and consume data while adhering to the IDS Information Model and interacting with external Identity Providers.

In addition, T2.4 investigate and leverage on Blockchain and smart contracts, to be used to certify transactions between multiple actors of the system.

The outcome of this task is the implementation of the Interoperable Data Space Framework, which include a set of interoperability services that can be deployed in local or cloud environments using various deployment approaches (centralized or distributed/federated). These services can be integrated with assets from other work packages, providing functionalities and services that project pilots can incorporate directly into their data collection and management pipelines.

1.3 Outline of the deliverable

Chapter 2 of the document focus on Platform Architecture, describing the system functionality and data models.

Chapter 3 analyses and describe the interfaces and communication mechanisms, exploring the API interfaces and user interface in detail.

Chapter 4 focus on languages, technologies and tools used by the connector.

Chapter 5 describes how to obtain, configure, and deploy data space connector services.

Finally, Chapter 6 concludes this document.

2 Data Space Overview and main concepts

2.1 IDS Connector

The Energy Dataspace Connector is based on the idea of IDS Connector but specialized for the Energy sector.

The IDS Connector is the core of the IDS Reference Architecture Model for implementing the concept of Data Space. Is the gateway to connect existing systems and their data to an IDS ecosystem. Its architecture and functionalities are defined by the IDS Reference Architecture Model (RAM) and specified by the certification criteria [1].

The IDS Connector allows to exchange data and enrich it with metadata. An important aspect of this are usage conditions, which can be defined, administrated, and implemented by the Connector. The metadata is described by the ontology of the IDS Information Model. The main advantage of the IDS reference architecture and the use of an IDS Connector is the decentralized data storage. This enables data integration from different data sources and allows data access exclusively through other IDS Connectors. Thus, a technical implementation of data sovereignty is guaranteed.

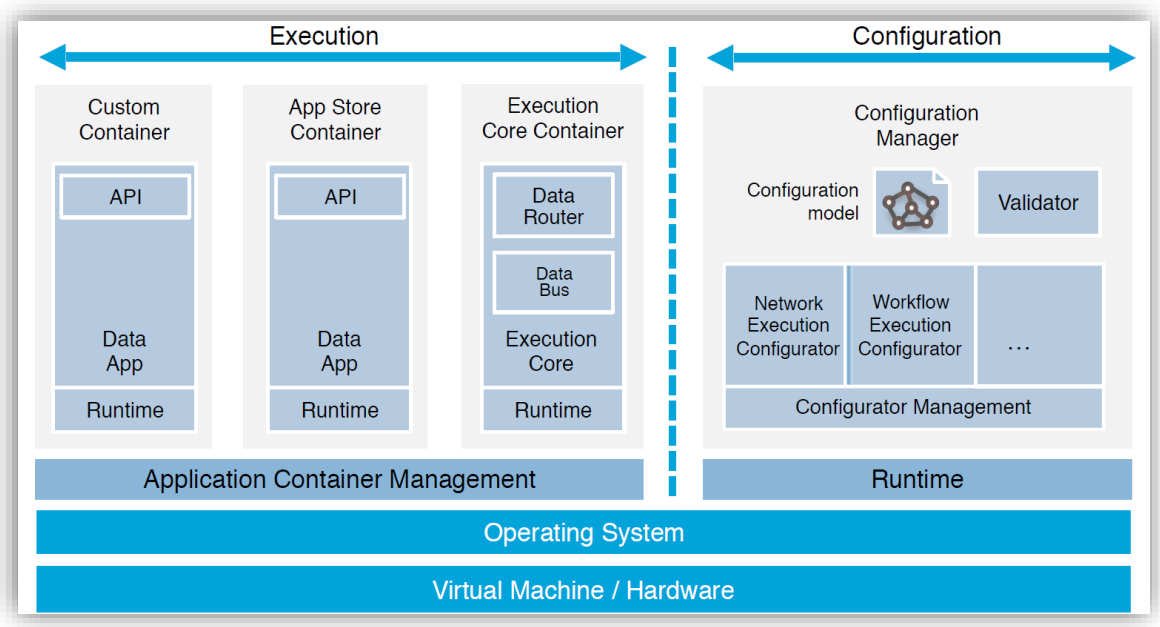

Figure 1: IDS Connector services diagram

An IDS Connector is composed of various system services:

- Execution core container with message systems (message router/bus)
- Configuration Manager to configure the Connector (execution core container, application container management, network, firewalls, etc.)
- Data Apps for data processing and handling
- Application container management
- Hardware/Operating system

The IDS Connector is the core part of a more complex architecture, designed to be modular and scalable, enabling a decentralised, secured and trusted data exchange among Data Space Participants.

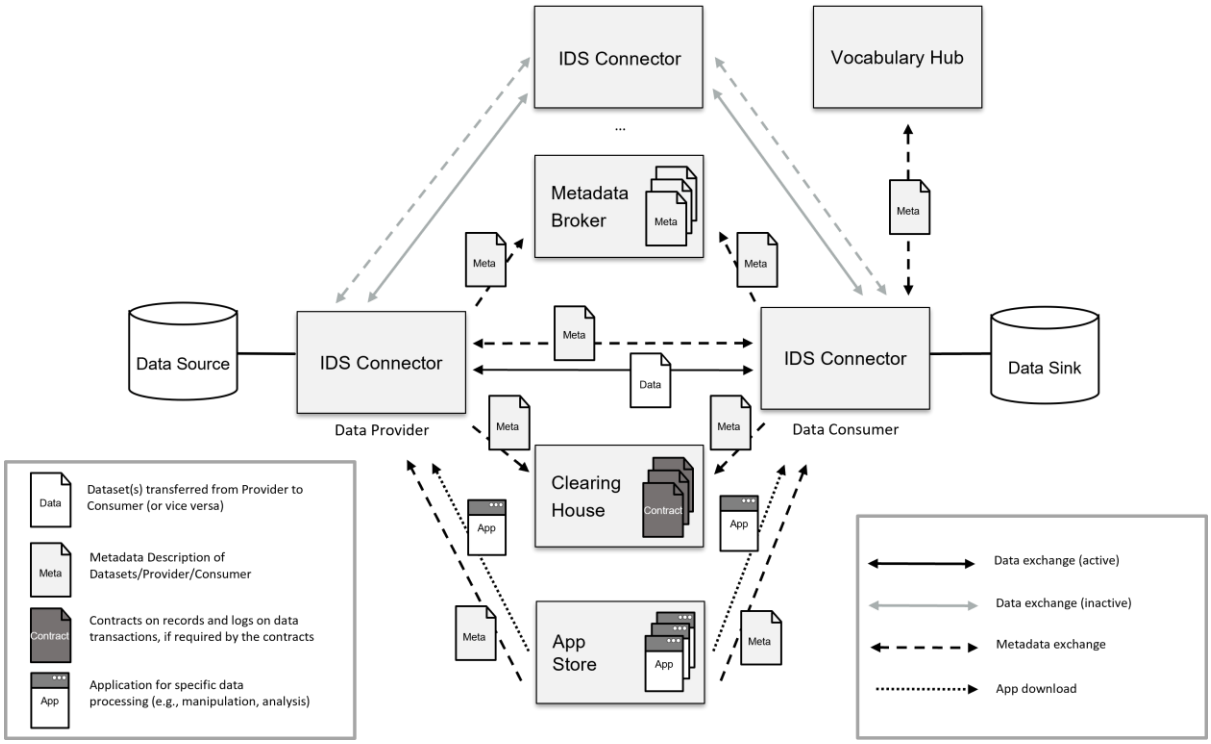

Figure 2: IDS RAM Components

The IDS Reference Architecture, shown in Figure 2, consists of the following core components:

- the Identity Provider
- the IDS Connector
- the App Store and Data Apps
- the Metadata Broker
- the Clearing House
- the Vocabulary Hub

2.2 TRUE Connector

TRUE connector is a connector for the IDS (International Data Space) ecosystem.

- It enables the trusted data exchange in order to be active part of an IDS Ecosystem, a virtual data space leveraging existing standards and technologies, as well as governance models well-accepted in the data economy, to facilitate secure and standardized data exchange and data linkage in a trusted business ecosystem.
- The connector is compliant with the latest IDS specifications and can be easily customized to fit a wide spread of scenarios thanks to the internal separation of Execution Core Container and Data App.

- It is integrable with a lot of existing IDS services and totally configurable in terms of internal/external data format (multipart/mixed, multipart/form, http-header) and protocols (HTTP, HTTPS, Web Socket over HTTPS, IDSCPv2).

Components:

- IDS Based ECC
- Configurable Data APP
- Clearing House
- UC APP

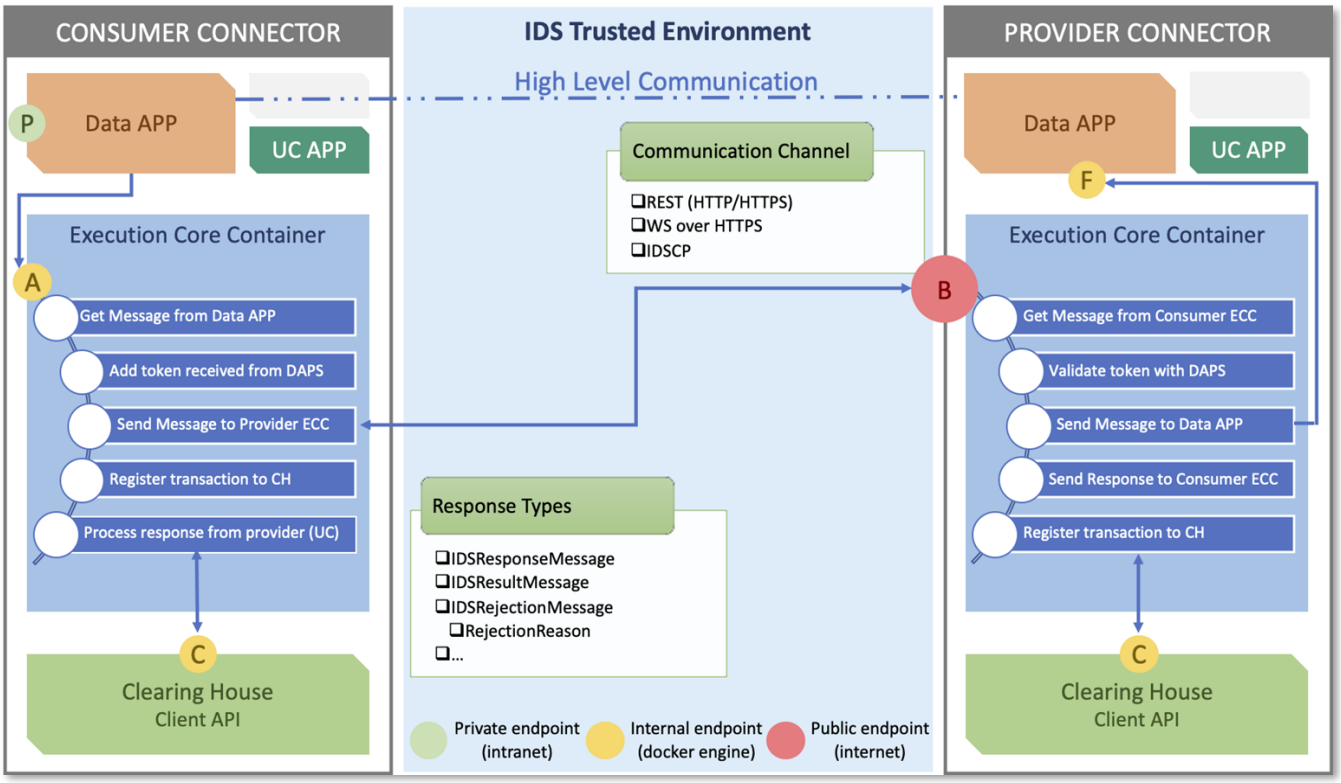

Figure 3: true connector diagram services

2.2.1 OneNet Connector and Energy Data Space

The True Connector architecture has been expanded within the OneNet project.

The following technologies have been added:

- FIWARE Context Broker fully integrated in the Connector Architecture
- Additional Data Services (Data Quality, Data Harmonisation…)
- Standardised API for any Energy Data

The OneNet connector has been customized for the energy domain, including:

- 10 services categories
- 64 energy services

- Standardization adoption (CIM [https://www.entsoe.eu/digital/common-information-model/cim-for-energy-markets/])

The following figure details the items that have been added.

Figure 4: OneNet energy standardization

2.3 Interoperable Data Space Framework

The architectural design of Interoperable Data Space Framework is structured to be modular, scalable and easily maintainable. This section outlines the key components, their interactions, and the overall design decisions that shape the system.

The Interoperable Data Space Framework mainly consists of two components: the Data Space Middleware and the Energy Data Space Connector.

The Figure 5 below shows the high-level architecture of the Interoperable Data Space Framework.

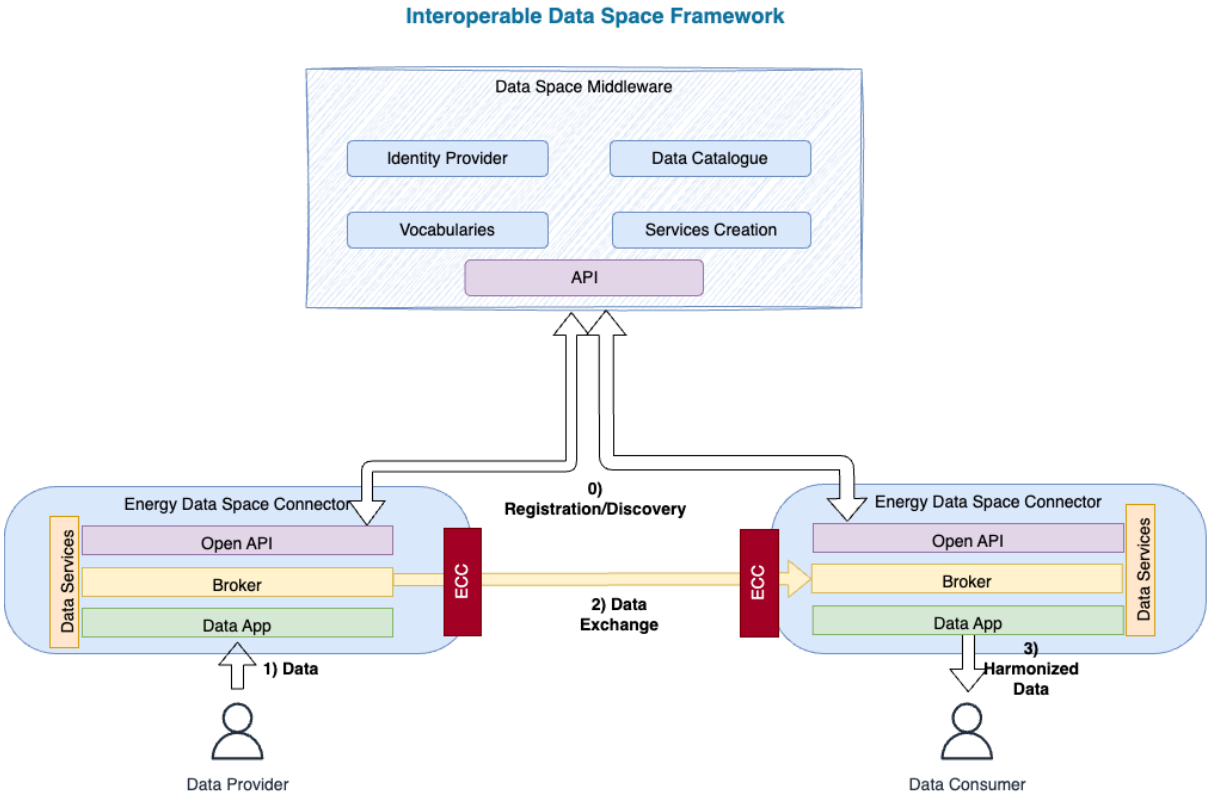

Figure 5: Interstore - Interoperable Data Space Framework

The Data Space Middleware implements all the central features of the Interoperable Data Space Framework including:

- Identity Management
- Vocabularies and Ontology for Energy Domain
- Data Catalogue discovery
- Services Creation

In this first version of the Interoperable Data Space Framework, the Data Space Middleware will be based on the OneNet Decentralized Middleware. In the next version, expected at March 2025, a new version of the Middleware will be implemented extending the existing one.

The Energy Data Space Connector is the core part of the Interoperable Data Space Framework. It extends the TRUE Connector and the OneNet Connector with additional

features and adaptation for the supporting the InterStore applications data generation and consumption.

The following figure shows the Energy Data Space Connector architecture.

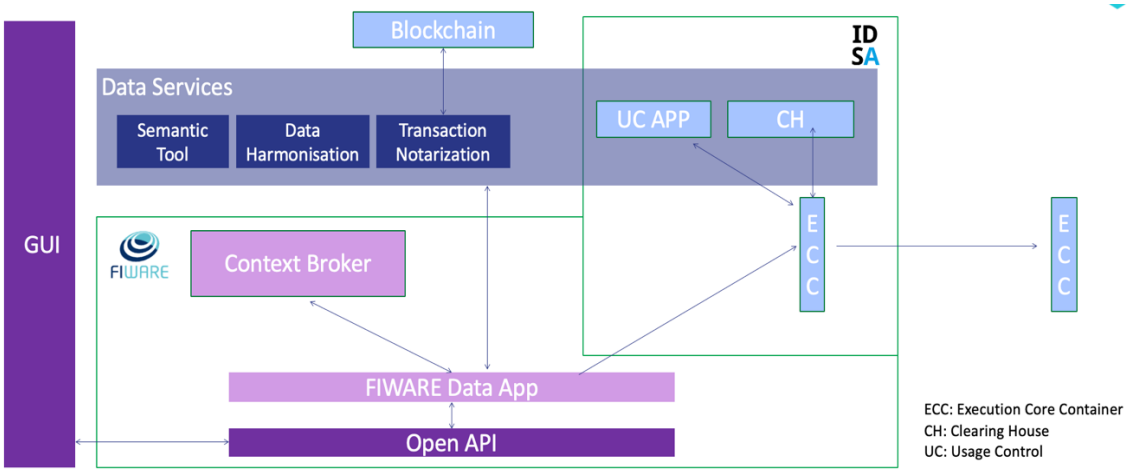

Figure 6: Energy Data Space Connector Architecture

IDS Components

IDS Components are implemented following the IDS specifications. Within the Energy Data Space Connector there are: Execution Core Container, Usage Control App and Clearing House.

The Execution Core Container plays the key role of communication and interaction with other connectors. It is able to establish the route with other connectors, providing metadata and information model and supporting several protocols.

The Usage Control App guarantees enforcement of data access and usage policies defined as part of the terms and conditions established when data resources or services are published or negotiated between providers and consumers. A data provider typically implements data access control mechanisms to prevent misuse of resources, while data usage control mechanisms are typically implemented on the data consumer side to prevent misuse of data. In complex data value chains, both mechanisms are combined by prosumers. Access control and usage control rely on identification and authentication.

Usage Control allow to define Usage Policies and Usage Enforcement. Each Data Provider can define usage control policies for their data, attached to the outbound data. Therefore, Data Space participants can be sure, that their data are treated according to their usage policies.

The Clearing House act as an intermediary that logs all activities performed during data exchange in the IDS ecosystem and it therefore provides clearing and settlement services for all financial and data exchange transactions.

Data in the Clearing House should be registered in an irrefutable way, in order to ensure the accounting access to and/or usage of data by different users. If dispute arises the Clearing House must be used as trusted third party to resolve this issue by providing secure and trusted logging system.

FIWARE Components

Within the Data Space Framework, FIWARE plays a key role for the standardization aspects and the Orion Context Broker represents a key service to ease the development and provisioning of smart and innovative applications that require context information and data stream management, processing and exploitation. The Orion Context Broker [2] is an HTTP Publish/Subscribe implementation—based on the NGSI standard—that enables management of the entire lifecycle of context information including updates, queries, registrations, and subscriptions. Orion allows defining a model of data (i.e., entity) to which publishers update values to be obtained by subscribers. Orion uses the NoSQL MongoDB database to store these entities and the last value recorded on them.

The FIWARE Data APP is a fully compliant FIWARE application, which support NGSI Standard for integrating Orion Context Broker functionalities in the Energy Data Space Connector through standardized open APIs.

FIWARE Data APP facilitates the sharing and exchange of data (i.e., data provision and data consumption/use) between data space participants.

FIWARE Data APP is strictly linked with the Context Broker and must be responsible for implementing the complete end-to-end data exchange process leveraging on the standards (e.g., NGSI-LD), as well as for offering standardized interfaces for the integration of the external platforms and systems through the Data Space Connector.

It should also address data format and data models specifications, establishing a common format for data model specifications and representation of data in data exchange payloads for ensuring fully interoperability among participants.

In addition, the FIWARE Data APP should also ensure data provenance and traceability, tracing and tracking all the process of data provision and data consumption/use. This feature is strictly linked with the Clearing House building block and provides the basis for the logging of transactions and accountability.

Open APIs

Open APIs are crucial for the integration of the Data Space Middleware services, as well for the integration of external platforms. The APIs are described more in details in Ch3.

Data Services

The Data Services module includes several services for supporting the data exchange within the Energy Data Space Connector.

First of all, it implements the possibility to select semantic vocabularies for the creation and consumption of data services as well the possibility to harmonize and translate the data exchanged. In the first version of the Connector, CIM is supported, while in the next version a complete support to IEEE2030.5 will be integrated.

In addition, the Data Transaction Services, based on blockchain technology and smart contracts, ensure the notarization of data transactions.

Graphical User Interface

This module facilitates the usage of the data space. It supports the discovery of data sources and data-driven services incorporating publishing and discovery mechanisms for data

resources and services, making use of common descriptions of resources, services, and participants. Such descriptions can be both domain-agnostic and domain-specific.

All these functionalities are implemented in a dedicated GUI of the Energy Data Space Connector, facilitating the discovery and publishing of data sources as well as the participation on the Data Space.

It also includes capabilities for publishing data resources in a standard and well recognised way, leveraging on the already know Data Catalogue Vocabulary provided by the OneNet Connector, so enabling an Energy Data Space.

Moreover, it allows to configure the data space environment, to create new data services and/or subscribe to them. Additional details about UI are provided in Ch.3.

2.4 Functionalities

In this first version the Energy Data Space Connector includes the following new features:

- Open Graphical User Interface
- Blockchain integration for data exchange transaction and verification

In the next version, it will also include:

- New data exchange mechanisms for service providers (e.g., Service Subscribe and Push data)
- Integration and support of new standards and protocols (e.g., IEEE 2030.5)

2.4.1 Open Graphical User Interface Module

The graphical interface has been completely rewritten trying to optimize the user experience and ease of use of the connector.

The starting point was the user interface already developed for the OneNet project.

The objective was to create a single page application that used modern technologies and was completely decoupled from the BackEnd API.

Furthermore, the new interface module did not have to have particular constraints on frameworks or proprietary libraries, so that it could be released under an open source license and used/evolved even after the end of the Interstore project.

The new interface includes the following features:

- User management and user authentication
- Dashboards
- Connector Settings
- Viewing the macro services of the Energy domain
- Creation and management of Services
- Subscription of services with acceptance workflow
- Data exchange (upload/download) on a service
- Timeline with visualization of smart contracts created on blockchain

In the next chapter all the technical information regarding the new user interface module will be included.

2.4.2 Transaction Notarization service

All transactions that take place within the Data APP call a blackchin-based transaction notarization service which takes care of verifying the data entity and writing some identifying transaction information in the blockchain infrastructure exploiting Smart Contracts implementation. In this way any transaction is tracked and verifiable.

2.4.2.1 Blockchain Technology for notarization

Blockchain technology has emerged as a disruptive innovation as the underlying technology for cryptocurrencies like Bitcoin and has developed with many applications beyond its original purpose.

At its core, a blockchain is a distributed ledger system designed to record and secure transactions across a network of computers. Unlike traditional centralized systems, a blockchain operates in a decentralized manner, utilizing a network of nodes (computers) to validate and record transactions.

The architecture of the blockchain technology leverages on the following key attributes:

1. *Decentralization*

One of the foundational principles of blockchain is decentralization. In contrast to centralized systems controlled by a single entity, blockchain networks are maintained by a distributed network of nodes. Each node has a copy of the entire blockchain ledger, ensuring that no single point of failure exists. This decentralization enhances security, resilience, and trust in the system.

2. *Consensus Mechanisms*

To validate and add transactions to the blockchain, a consensus mechanism is employed. The most common mechanisms include Proof of Work (PoW) and Proof of Stake (PoS). PoW requires miners to solve complex mathematical puzzles to validate transactions, while PoS relies on validators who "stake" cryptocurrency as collateral to confirm transactions. These mechanisms ensure that only legitimate transactions are added to the ledger.

3. *Immutability*

Once a transaction is recorded on the blockchain, it becomes extremely challenging to alter or delete. The immutability of blockchain data is a fundamental feature that provides a high level of security and trust. It is achieved through cryptographic hashing, where transactions are grouped into blocks, and each block is linked to the previous one.

4. *Transparency*

Blockchains are open and transparent by design. Anyone can view the entire transaction history, promoting trust and accountability. This transparency is particularly valuable in applications where trust is critical, such as supply chain management and financial services.

Blockchain and Smart Contract – Ethereum

Ethereum, often referred to as "Blockchain 2.0," is a programmable blockchain that builds upon the principles introduced by Bitcoin and extends the capabilities of blockchain technology, introducing Smart Contracts.

Ethereum was conceptualized and developed by Vitalik Buterin in 2014 [3] , and the network officially launched on July 30, 2015. The genesis of Ethereum was rooted in addressing perceived limitations in Bitcoin. While Bitcoin primarily aimed to create a digital currency, Ethereum sought to provide a more versatile platform for Decentralized Applications (DApps) and smart contracts.

The main differences from Bitcoin are:

1. *Purpose and Functionality*

Bitcoin was created as a digital currency and a store of value, focusing primarily on peer-to-peer electronic cash transactions. Ethereum, on the other hand, was designed as a general-purpose blockchain platform that enables developers to build and deploy DApps and smart contracts.

2. *Smart Contracts*

Ethereum's most important features is its support for smart contracts. Smart contracts are self-executing contracts with the terms of the agreement directly encoded into code. Bitcoin, while programmable to an extent, was not designed with the same level of flexibility as Ethereum. Smart contracts on Ethereum enable a wide range of applications, from Decentralized Finance (DeFi) to supply chain management and more.

3. *Turing Completeness*

Ethereum's programming language, Solidity, is Turing complete, which means it can perform any computation that can be expressed algorithmically. Bitcoin's scripting language is intentionally limited to prevent certain types of potentially harmful or resource-intensive computations. Ethereum's Turing completeness allows for a broader range of applications and complexity in DApps and smart contracts.

Ethereum's introduction of DApps opened the door to a wide range of decentralized applications. DApps operate on the Ethereum blockchain, utilizing smart contracts to automate processes and transactions. This has led to the emergence of a vibrant ecosystem of applications, including DeFi platforms, NFT marketplaces, and decentralized governance systems.

Despite its numerous achievements, Ethereum has faced challenges, particularly in terms of scalability and high gas fees. The increasing demand for Ethereum's services led to congestion on the network, resulting in higher transaction costs. These challenges prompted the development of Layer 2 scaling solutions like Polygon to alleviate these issues.

In order to improve the scalability, energy efficiency, and security concerns associated with the original PoW consensus mechanism, Ethereum has been undergoing a significant upgrade known as Ethereum 2.0. In this second version, Ethereum is transitioning to a PoS consensus mechanism, where validators are chosen to create new blocks based on the amount of cryptocurrency they hold and are willing to "stake" as collateral. This transition is expected to improve network efficiency and reduce its carbon footprint.

Layer 1 vs Layer 2 – Polygon

Ethereum and Bitcoin are considered Layer 1 blockchains. Layer 1 refers to the primary blockchain layer where the core network operates, transactions are processed, and security is maintained. Layer 2, on the other hand, is a secondary layer built on top of Layer 1, designed

to enhance scalability and reduce transaction costs by processing transactions off the main blockchain while benefiting from its security.

A layer 2 refers to any off-chain network, system, or technology built on top of a Layer 1 blockchain that helps extend the capabilities of the underlying base layer network. Layer-2 networks can support any blockchain to introduce enhancements such as higher transaction throughputs [4].
One core requirement for a network, system, or technology to be considered a layer 2 is that it inherits the security of the blockchain it is built on top of. Transaction data must be verified and confirmed by the underlying blockchain network rather than a separate set of nodes.
For blockchains that sacrifice scalability to achieve higher decentralization and security, Layer 2 enable greater transaction throughput, which can lead to lower fees. Layer 2 can be seen as one solution to the problem of scalability, enabling fast and scalable execution without compromising on decentralization or security. [4].

The main Layer 2 blockchain built on top of Ethereum is Polygon [5].
Polygon provides a framework for creating Ethereum-compatible blockchains which process transactions more quickly and at a lower cost compared to the Ethereum main net. Polygon's MATIC token is central to its ecosystem, serving various purposes, including paying transaction fees, and participating in network governance.
Polygon has gained prominence in the blockchain space due to its utility in a wide range of use cases.

Layer 1 and Layer 2 blockchain solutions offer distinct advantages and trade-offs:

Layer 1 Advantages

- Provide robust security and decentralization.
- Serve as the foundation for the entire blockchain ecosystem, hosting critical smart contracts and assets.
- Highly secure, backed by a large network of miners and validators.

Layer 2 Advantages

- Alleviate scalability issues by processing transactions off the main blockchain.
- Reduce transaction costs, making blockchain applications more accessible.
- can experiment with innovative consensus mechanisms, such as PoS, to improve efficiency.

2.4.2.2 Blockchain Notarization Service implementation

The Blockchain Notarization Service is activated at any data exchange. An example of data exchanged flow between data provider and data consumer is described below.

1. In the first step data provider create a new service.
2. Then consumer request to subscribe that service.
3. Data provider accept the request from consumer and publish new data.
4. The consumer is able to consume the data.
5. At this point, the blockchain notarization service is called and the transaction is notarized into blockchain. A new smart contract is deployed with core metadata and data content hashing.
6. It's possible to verify the transaction with any blockchain explorer. In addition to the metadata, we can also see the address of the smart contract.

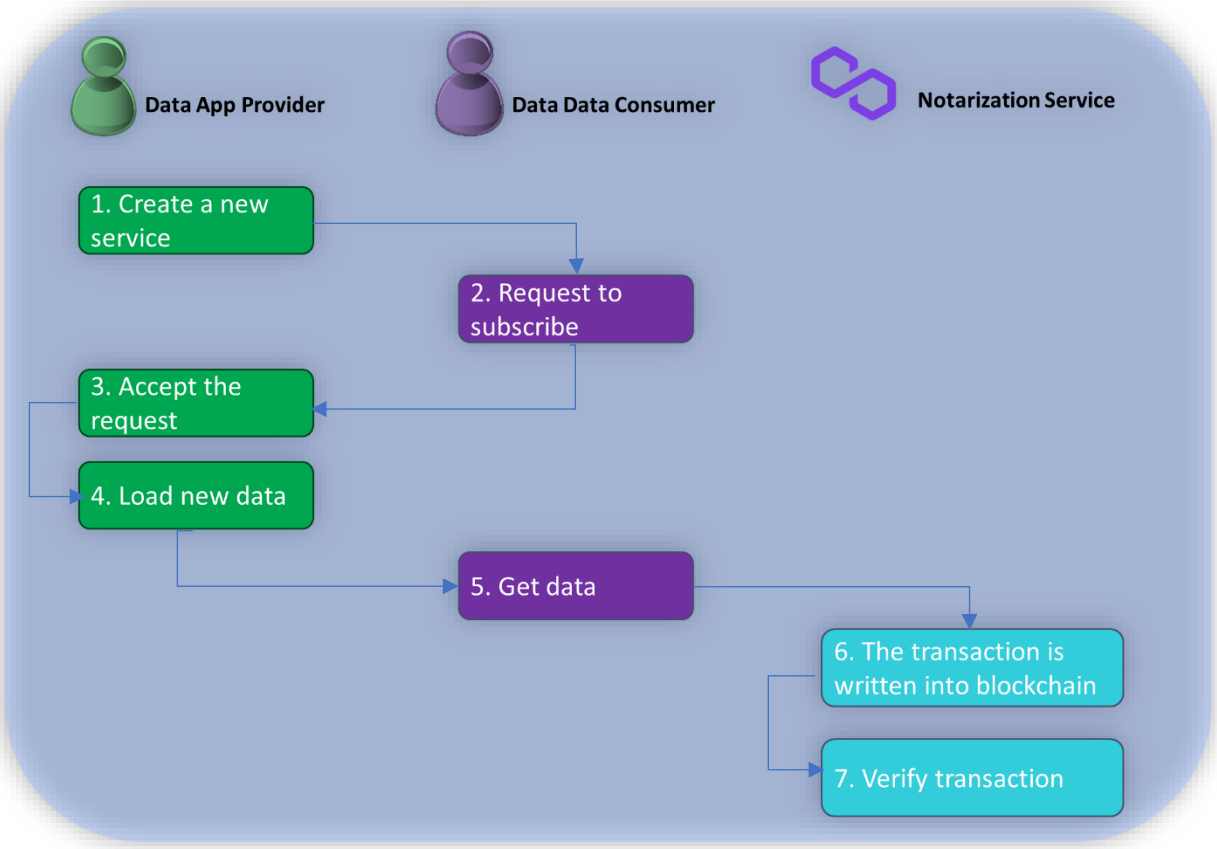

Figure 7: An example of flow with Notarization Service

2.4.3 Additional Functionalities expected for the final version

2.4.3.1 Service Subscribe and Push data scenario

The new feature allows to manage additional scenarios, in which a service provider creates a service and one or more data providers can load data into the service.

The expected steps are as follows:

1. The Service Provider create a new service, implement a Rest Web Service, according to a well defined interface, and specifies the URL of the service to be invoked (in the Service creation GUI of Connector)
2. Data providers subscribe to the service, and after request has been accepted from service provider, can push data (content and metadata) using the connector in a classic way
3. As part of the loading process, the web service provided in the configuration (and exposed by the service provider) is called, passing both the content file and the metadata. In this way the data is immediately made available to the service provider (outside the connector)

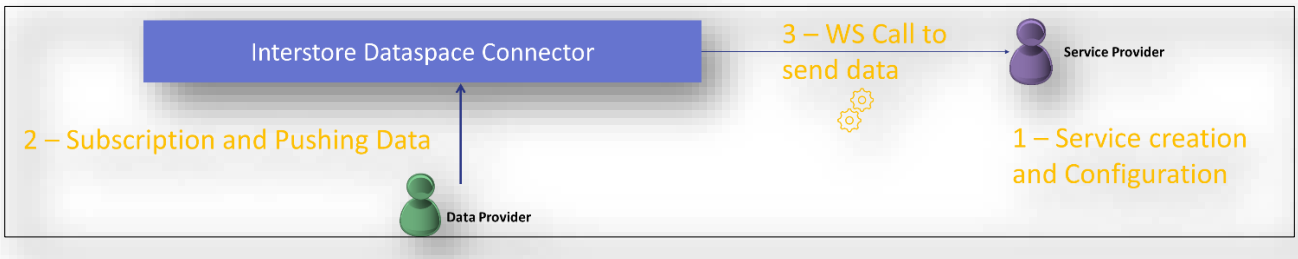

Figure 8: Push data scenario

2.4.3.2 Integration and support of new standards and protocols (IEEE2030.5)

The new Energy Data Space connector plans to implement a new interface adhering to the IEEE2030.5 standard in order to allow direct integration with devices or software modules that support these standards.

This integration will be implemented in the next version of the Data Space Connector, including the other open tools developed in T2.1 and T2.2.

2.5 Data Models

The data models follow the Open API specification [6].

2.5.1 User

Table 1: User Model

Field	Type	Description
username	String	Required Username of the user logged in
password	String	Required Password of the user logged in
email	string	User's email provided during the registration

2.5.2 Connector settings

Table 2: Connector settings model

Field	Type	Description
ecc_url	String	Execution Core Container URL
name	String	Name of the company
Id	String	Id of the user logged in
Email	String	Email of the user logged in
ed_api_url	String	Local API URL
username	String	Username of the user logged in
data_app_url	String	Data app URL

2.5.3 Cross platform services

Table 3: Cross platform services model

Field	Type	Description
Category	String	Category of the cross-platform service
Service	String	Name of the cross-platform service
Businnes object name	String	Name of the business object
Businnes object code	String	Code of the business object
Service description	String	Description provided for the service
Profile selector	String	Format of the file (e.g. csv, json ecc..)
profile_description	string	Description of the format
Cross platform service id	String	Id of the cross-platform service
file_schema	String	File schema providing the format
file_schema_sample	String	Example of file schema providing the format
file_schema_filename	String	Name of the file schema providing the format
file_schema_sample_filename	String	Name of file_schema_sample

data_catalog_service	data_catalog_service object	Information about the service

2.5.4 My offered services

Table 4: Offered services model

Field	Type	Description
category	String	Category of the offering service
title	String	Required Title of the offering service
created_on	String	Creation date of the offering service
profile_selector	String	Format of the file (e.g. csv, json ecc..)
profile_description	String	Description of the format
status	String	Status of the service (active | disabled)
subscriptions	String	User subscribed the service
comments	String	Comments from the service applicant
id	String	Id of the offered service
data_catalog_service	data_catalog_service object	Information about the service object
file_schema	String	File schema providing the format
file_schema_sample	String	Example of file schema providing the format
file_schema_filename	String	Name of the file schema providing the format
file_schema_sample_filename	String	Name of file_schema_sample
active_from	String	Activation date dd/mm/yyyy hh:mm
active_to	String	Activated until this date dd/mm/yyyy hh:mm
data_catalog_business_object	data_catalog_business_object object	Business object information

2.5.5 My subscriptions

Table 5: Subscriptions model

Field	Type	Description
category	String	Category of the offering service
title	String	Title of the offering service
user_offering	String	User offering the service
status	String	Status of the service

created_on	String	Creation date of the offering service
comments	String	Comments for the subscriptions
profile_selector	String	Format of the file (e.g. csv, json ecc..)
profile_description	String	Format description
data_catalog_data_offering_id	String	Id of the service provided
data_catalog_data_offerings	data_catalog_data_offerings object	Information on the service provided

2.5.6 Requests on offered services

Table 6: Requests model

Field	Type	Description
category	String	Category of the offering service
title	String	Title of the offering service
user_offering	String	User offering the service
status	String	Status of the service
created_on	String	Creation date of the offering service
comments	String	Comments for the subscriptions
category	String	Category of the offering service
title	String	Title of the offering service
modified_on	String	Date when the dd/mm/yyyy hh:mm
user_requesting	user_requesting object	Information abou User requesting
data_catalog_data_offerings	data_catalog_data_offerings object	Data offerings information
id	string	Id of the request
user	user object	User that is accepting/rejecting information

2.5.7 Data provided

Table 7: Data provided model

Field	Type	Description
data_catalog_category_name	String	Name of the data category
id	String	Data id
title	String	Required Data title
created_on	String	Creation date of the offering service dd/mm/yyyy hh:mm
description	String	Description of the data

fileName	String	Name of the file
created_by	String	User who creates the data
data_catalog_data_offerings_id	String	Required Service id for new data
message	String	Required String containing the file (Base64)
profile_selector	String	Format of the file (e.g. csv, json ecc..)
profile_description	String	Format description
category	String	Category of the offering service
title	String	Title of the offering service

2.5.8 Data consumed

Table 8: Data consumed model

Field	Type	Description
category	String	Category of the offering service
user_offering	String	User offering the service
created_on	String	Creation date of the offering service dd/mm/yyyy hh:mm
title	String	Data title
description	String	Description of the data
filedata	String	String containing the file (Base64)
fileName	String	Name of the file
data_catalog_data_offerings_id	String	Service id for new data
data_catalog_data_offerings	data_catalog_data_offerings object	Information about the service for new data

2.5.9 Timeline

Table 9: Timeline model

Field	Type	Description
left_side	number	If it is equal to 0 the item rapresents consumed data, if it is equal to 1 provided data
isTheLastPage	number	If it is equal to 0 is the last page and is not possible to expand anymore
description	String	Idem description (e.g category and file name)
title	String	Item title
nav	String	String contains the data id
data_app_url	String	Contains target port and protocol for the "/data-app-consumer/registration" api
ecc_url	String	Execution Core Container URL

3 Interfaces and Communication Mechanisms

3.1 API Interfaces

All modules of the Energy Data Space Connector communicate with each other mainly using Rest APIs.

APIs can be divided into the following 2 groups:

- Centralized APIs
- Decentralized or local connector APIs

3.1.1 Centralized APIs

This group of APIs is used for all operations involving:

- Users (Authentication, Info and Connector Settings)
- Services Management
- Data exchanges

In the next paragraphs the APIs will be detailed.

3.1.1.1 Users (Authentication, Info and Connector Settings)

Table 10: Users APIs

Name	Url	Method	Parameters	Responses
User Authentication	/api/user/auth	POST	In body: username : String password : String	Success (200) accessToken: String refreshToken String user: {User} Error (XXX) Error code - *String* Error Message - *String*
User Info	/api/user/current	GET	-	Success (200) user: {User} Error (XXX) Error code - *String* Error Message - *String*
Get Connector Settings	/api/custom-query/data-objects/?id=e48046c9-0b94-41d2-9ad4-206f1604b821	GET	In Request: Id: String	Success (200) Connector settings info Error (XXX) Error code - *String* Error Message - *String*
Save Connector Settings	/api/custom-query/data-objects/?id=e48046c9-0b94-41d2-9ad4-206f1604b821	POST	In Request: Id: String ed_api_url: String data_app_url: String ecc_url: String	Success (200) Error (XXX) Error code - String Error Message - String

3.1.1.2 Services Management

Table 11: Services Management APIs

Name	Url	Method	Parameters	Responses
Cross Platform Service List	/api/datalist/cross_platform_service/page/0	GET	In Path: page : int In Request: <filters>: string	Success (200) listContent: Array[CrossPlatformService] Error (XXX) Error code - *String* Error Message - *String*
Cross Platform Service Get by id	api/dataset/cross_platform_service/{{entity_id}}	GET	In Path: entity_id: string	Success (200) "data_catalog_business_object": {CrossPlatformService} Error (XXX) Error code - *String* Error Message - *String*
Offered Services List	/api/datalist/my_offered_services/page/0	GET	In Path: page: int In Request: <filters>: string	Success (200) listContent: Array[OfferedService] Error (XXX) Error code - String Error Message - String
Offered Services Get By Id	/api/dataset/my_offered_services/{{entity_id}}	GET	In Path: entity_id: string	Success (200) "data_catalog_business_object": {OfferedService} Error (XXX) Error code - *String* Error Message - *String*
Offered Services Create/Update	/api/dataset/my_offered_services	POST	In Body: data_catalog_data_offerings: {OfferedService}	Success (200) id of { OfferedService} Error (XXX) Error code - String Error Message - String
Subscription List	/api/datalist/my_subscriptions/page/0	GET	In Path: page: int In Request: <filters>: string	Success (200) listContent: Array[Susbscription] Error (XXX) Error code - String Error Message – String
Subscription Get By Id	/api/dataset/my_subscriptions/{{entity_id}}	GET	In Path: entity_id: string	Success (200) "data_catalog_data_requests": {Subscription} Error (XXX) Error code - String Error Message - String
Subscription Create	/api/dataset/my_subscriptions	POST	In Body: data_catalog_data_requests: {Subscription}	Success (200) id of {Subscription} Error (XXX) Error code - String Error Message – String

Request List	/api/datalist/requests_on_offered_services/page/0	GET	In Path: page: int In Request: <filters>: string	Success (200) listContent: Array[Request] Error (XXX) Error code - String Error Message – String
Request Get By Id	/api/dataset/requests_on_offered_services/{{entity_id}}	GET	In Path: entity_id: string	Success (200) “data_catalog_data_requests”: {Request} Error (XXX) Error code - String Error Message - String
Request Create	/api/dataset/requests_on_offered_services	POST	In Body: data_catalog_data_requests: {Request}	Success (200) id of {Request} Error (XXX) Error code - String Error Message – String

3.1.1.3 Data exchanges

Table 12: Data Exchanges APIs

Name	Url	Method	Parameters	Responses
Data Provided List	/api/datalist/data_provided/page/0	GET	In Path: page: int In Request: <filters>: string	Success (200) listContent: Array[Data Provided] Error (XXX) Error code - String Error Message – String
Data Provided Get By Id	/api/dataset/data_provided/{{entity_id}}	GET	In Path: entity_id: string	Success (200) “data_send”: {Data Provided } Error (XXX) Error code - String Error Message - String
Data Provided Create	/api/dataset/data_provided	POST	In Body: Data_send: {DataProvided}	Success (200) id of {DataProvided} Error (XXX) Error code - String Error Message – String
Data Consumed	/api/datalist/data_consumed	GET	In Path: page: int In Request: <filters>: string	Success (200) listContent: Array[Data Provided] Error (XXX) Error code - String Error Message – String
Data Consumed Get By Id	/api/dataset/data_consumed/{{entity_id}}	GET	In Path: entity_id: string	Success (200) “data_send”: {Data Consumed } Error (XXX) Error code - String Error Message – String

Timeline List	/api/timeline/data/?id=d6342c52-8995-4a0d-b42d-894ffc600a3d&enabled=1	GET	In Request: created_after: data created_before: data currentPage: int	Success (200) resultList: Array[DataEntity] Error (XXX) Error code - String Error Message – String

3.1.2 Data App APIs

Data App APIs are used to manage entities within the data space connector.
In the next table the APIs will be detailed.

Table 13: Data App APIs

Name	Url	Method	Parameters	Responses	Description

Create Entity	/createentity/	POST	In Body:
			{
			Id: String
			Type: String
			Category: Object
			Address: Object
			Location: Object
			Name: Object
			}
				Success (200)
				
				Error (XXX)
					Used to create a new entity and upload content (metadata and file) to data space connector

Get Entity	/getentity/{{entity_id}}	GET	In Body:
			{
			entityId: String
			eccUrl: String
			}
				Success (200)
				{
				Id: String
				Type: String
				Category: Object
				Address: Object
				Location: Object
				Name: Object
				}
				Error (XXX)
					Used to get an entity (metadata and file) from data space connector

Registration	/registration	POST	In Body:
			{
			entityId: String
			eccUrl: String
			}
				Success (200)
				{
				“pid”: String,
				“smartContract Address”: String
				}
				
				Error (XXX)
					Used to manage (create or get) a registration within data space connector. A registration is a sort of contract between provider and consumer.

3.2 UI Interfaces

This chapter presents in detail the graphical interfaces of the Energy Data Space Connector.

3.2.1 Login

The Login page allows access to the Energy Data Space connector web platform for all users of the system. The module transmits the credentials to the back end server and, if these are correct the login succeed, otherwise an error is shown to the user.

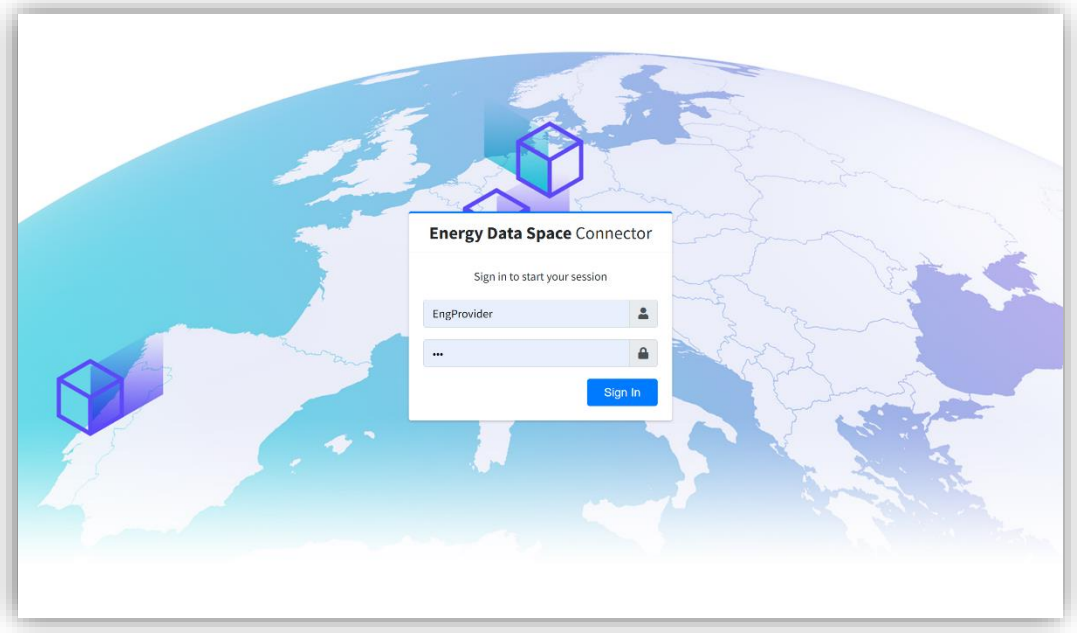

Figure 9: Login Page

3.2.2 Connector Settings

The connector settings summarizes the information of the connected user and allows you to change the connector parameters and the API to use.

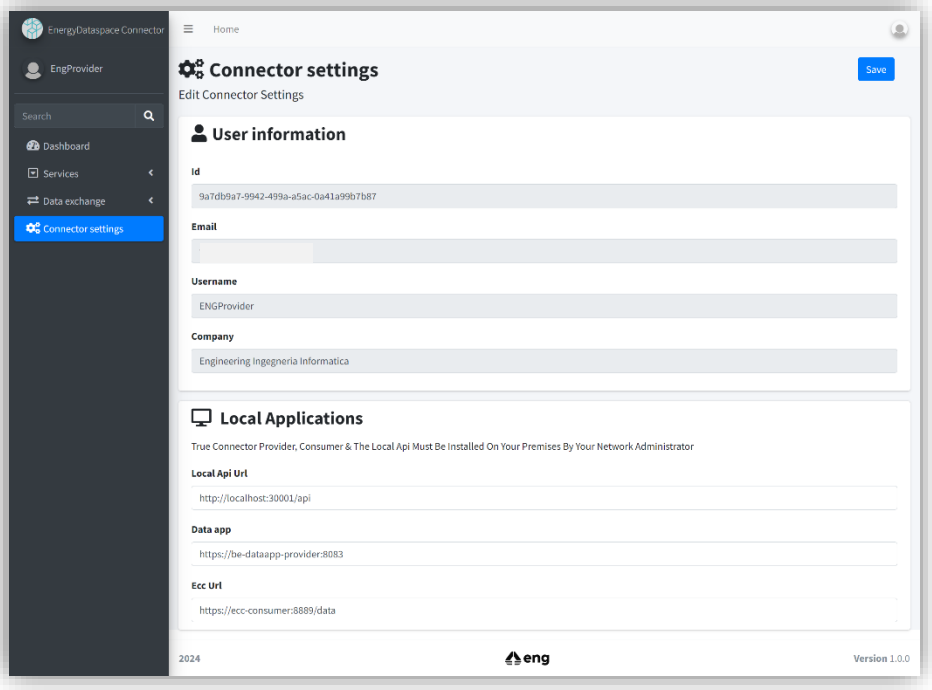

Figure 10: Connector Settings

3.2.3 Dashboard

The dashboard of the new user interface provides some widgets with the main information about user and connector (e.g. data provided, data consumed, offered services, subscriptions).

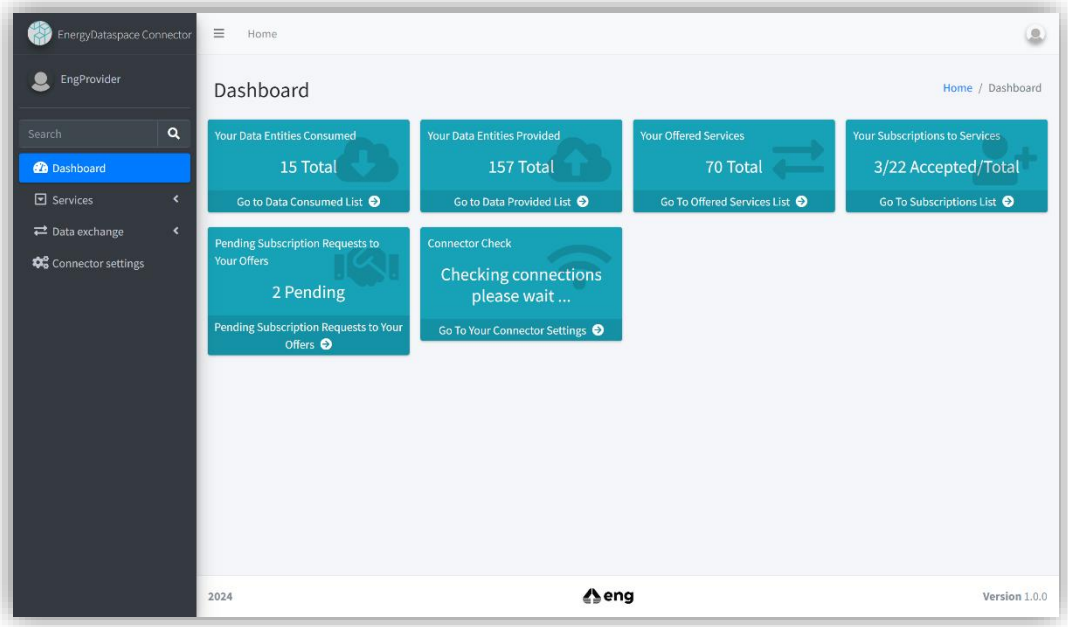

Figure 11: Dashboard

3.2.4 Cross platform services

This interface shows the list of all cross-platform services, with the possibility of filter by category or specific field. The lists is paginated.

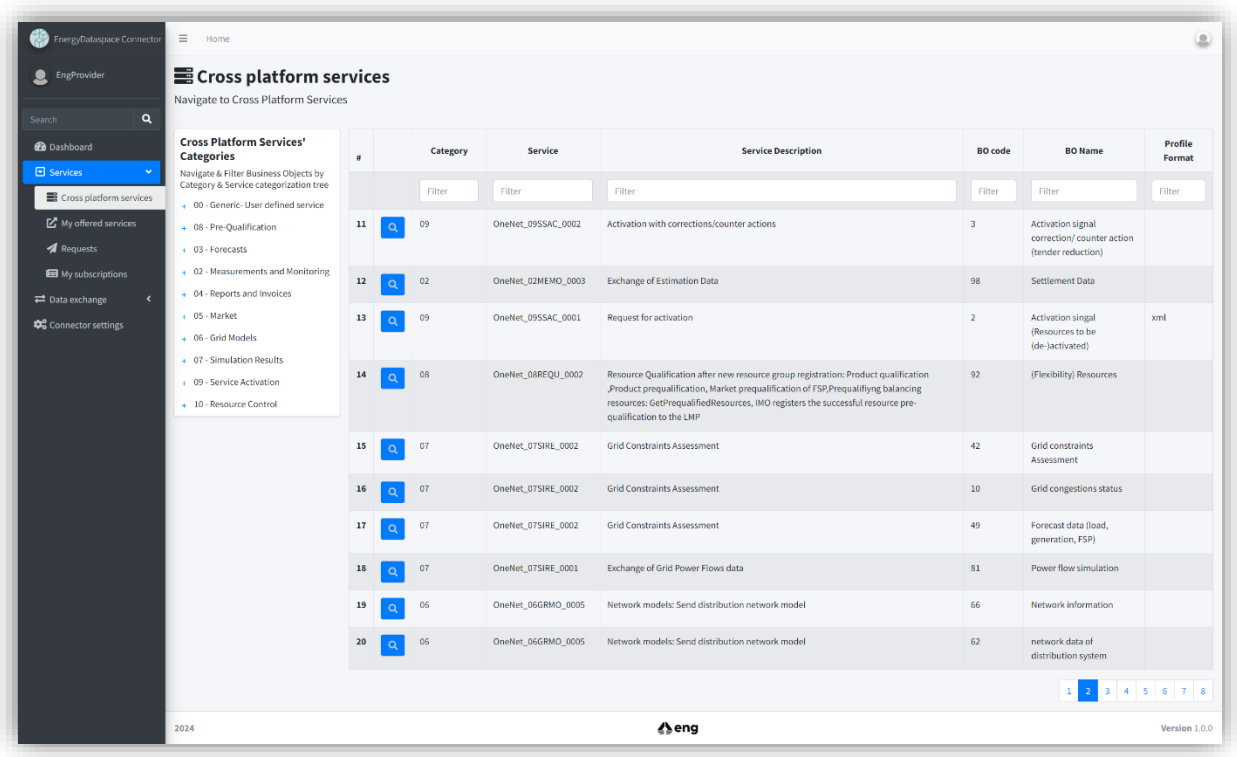

Figure 12: Cross Platform Services

3.2.5 Offered Services

The offered services list shows all services created by the user. It's possible always use filters and create new services using the new button. We have a detail button in the table and the ability to see any subscription requests.

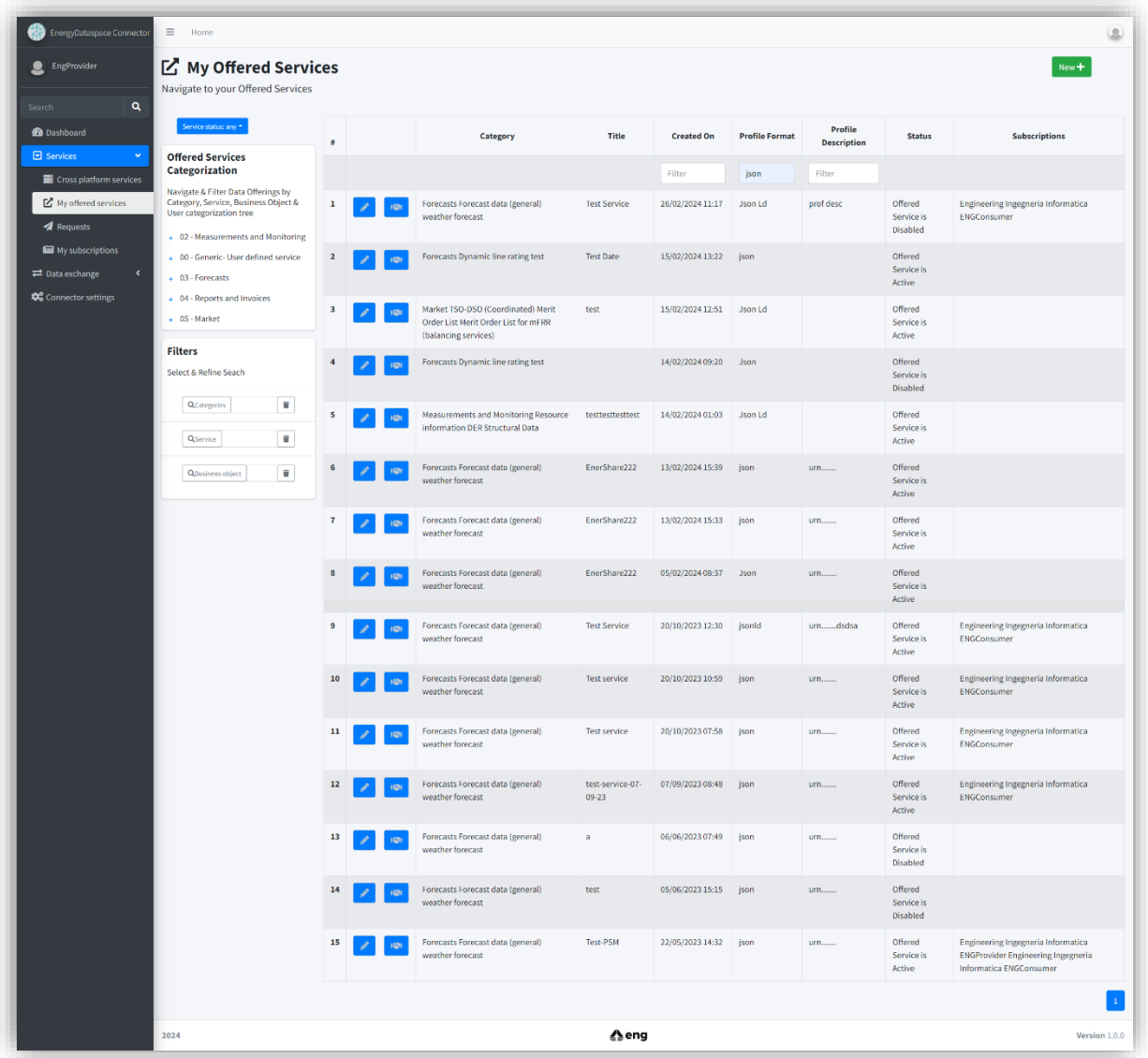

Figure 13: Offered Services

3.2.6 Requests

The request page shows all request received by the user. The user can enter in detail page and accept or reject the request.

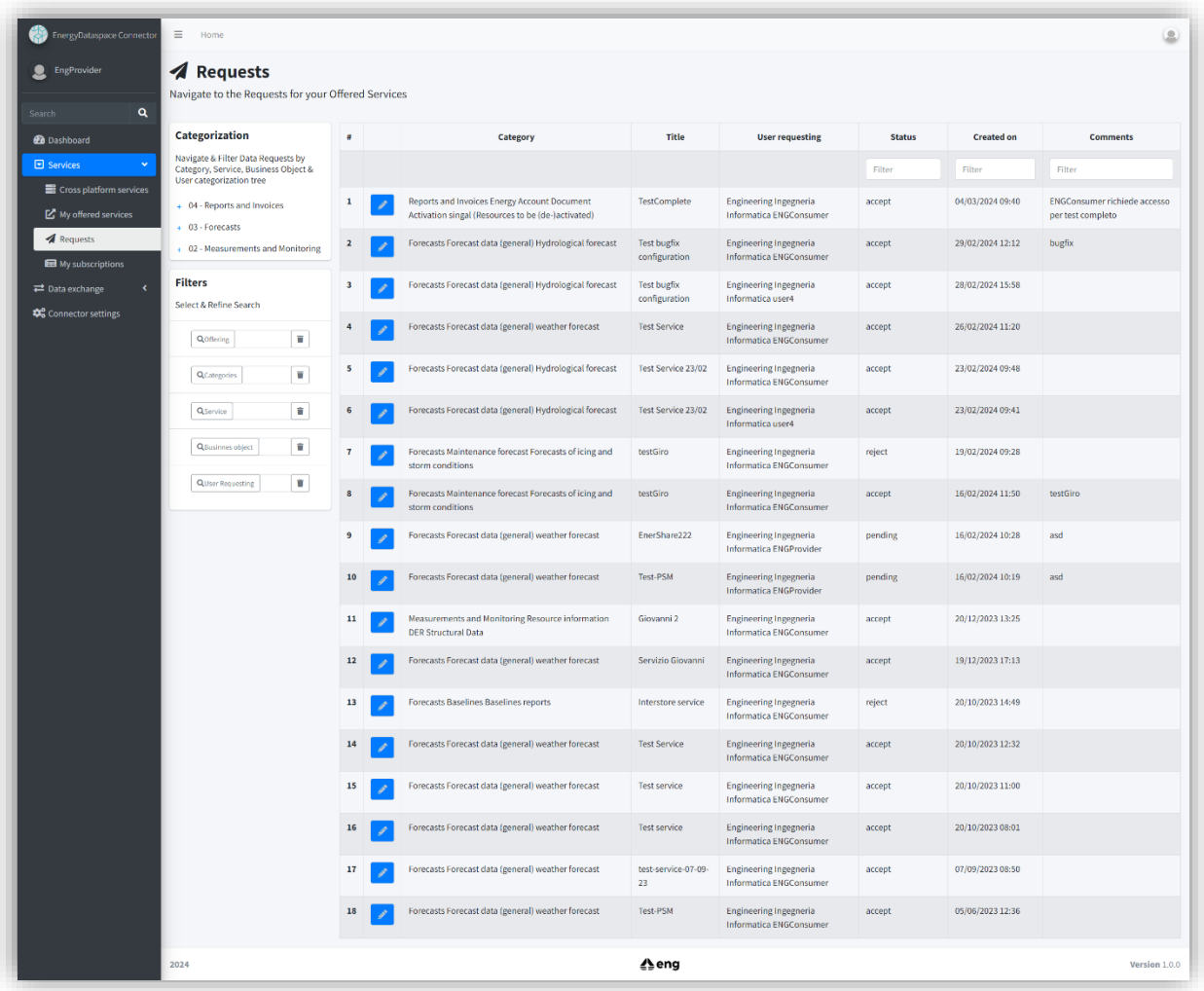

Figure 14: Requests

3.2.7 My Subscriptions

The My Subscriptions interface allows users to view the list of personal requests to sign up for services. The New subscriptions button opens a page for entering a new subscription request for a service.

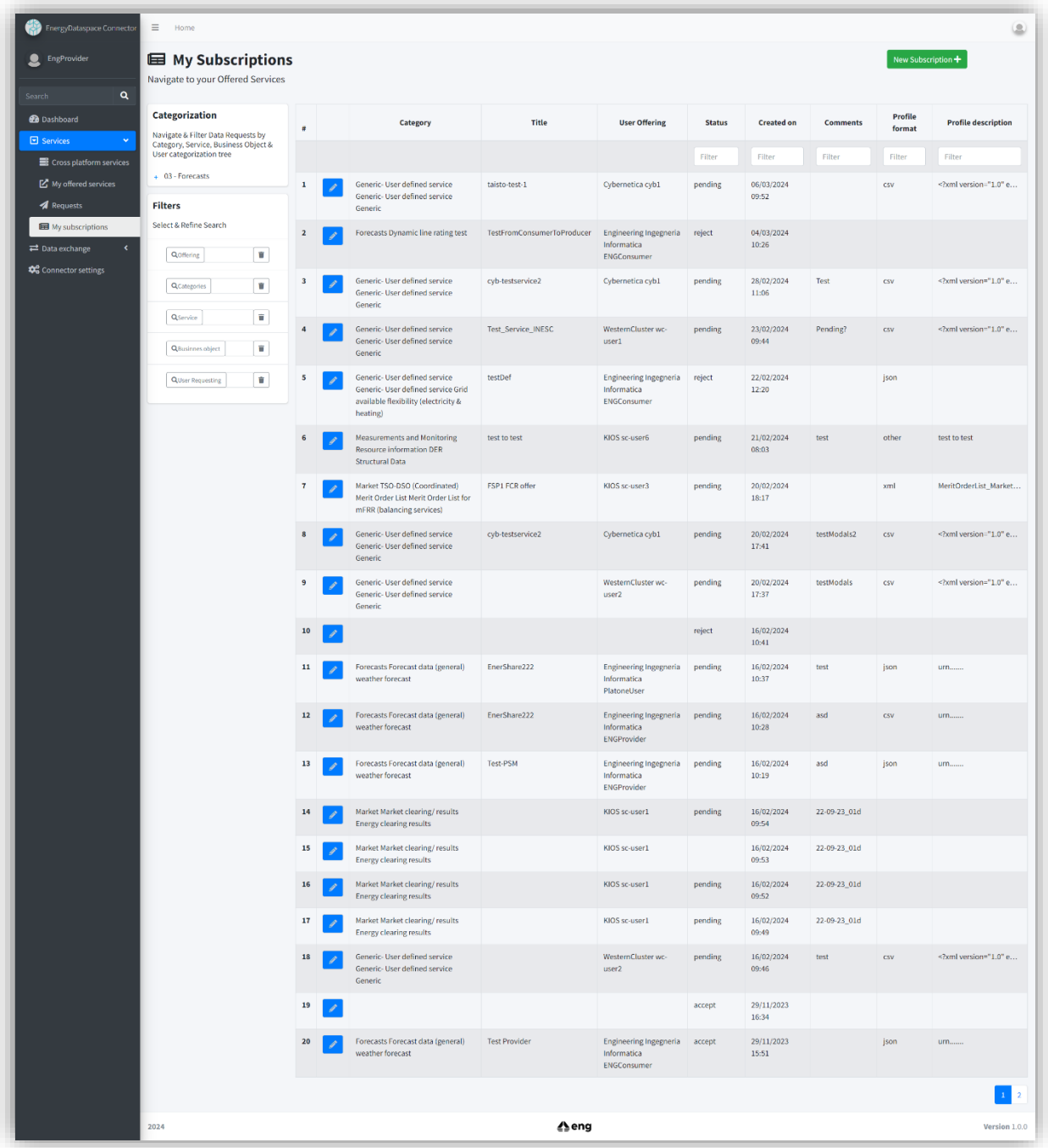

Figure 15: My Subscription

3.2.8 Provide Data

The provide data list shows all created data. The new button opens a form with the possibility to create a new data and upload a document or file.

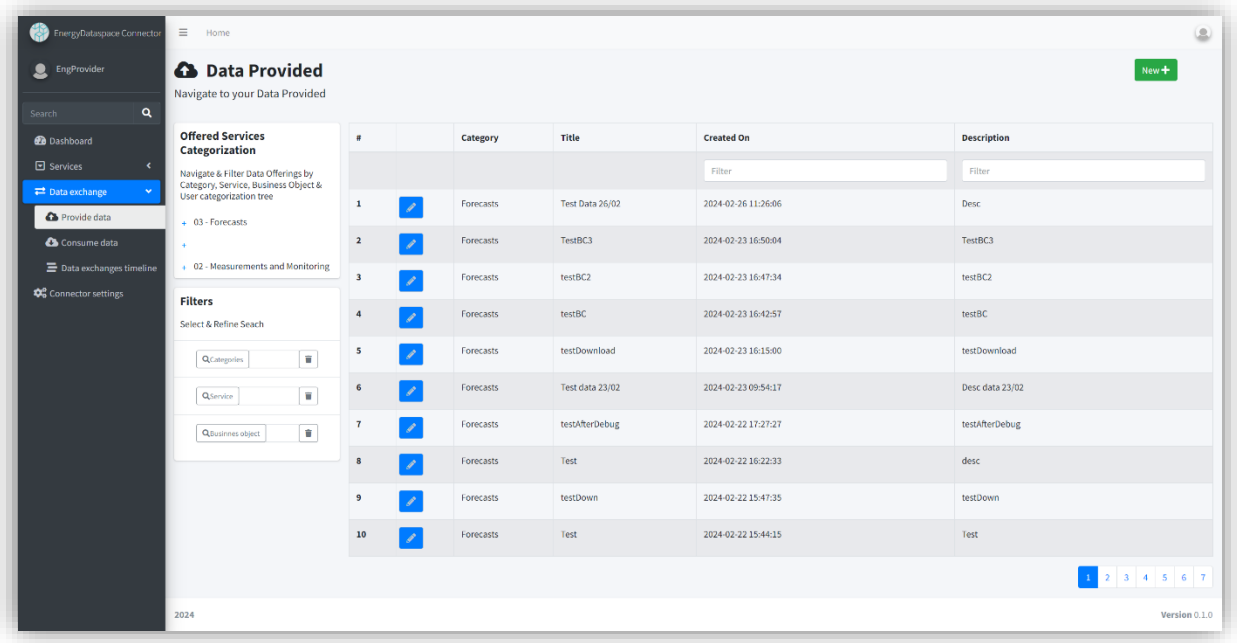

Figure 16: Provide Data List

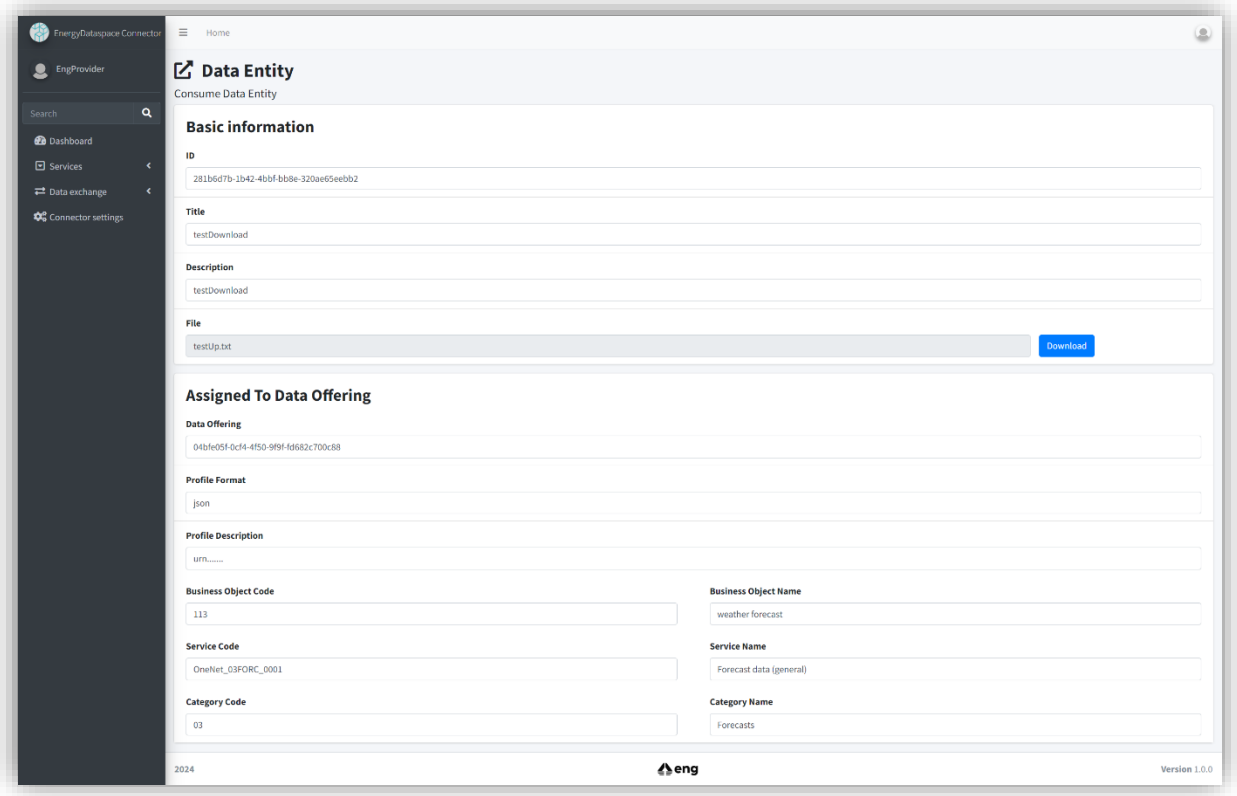

Figure 17: Provide Data Form

3.2.9 Consume Data

This is the consume data list interface. By clicking on a table row(lens symbol) you can access the data entity and download the document.

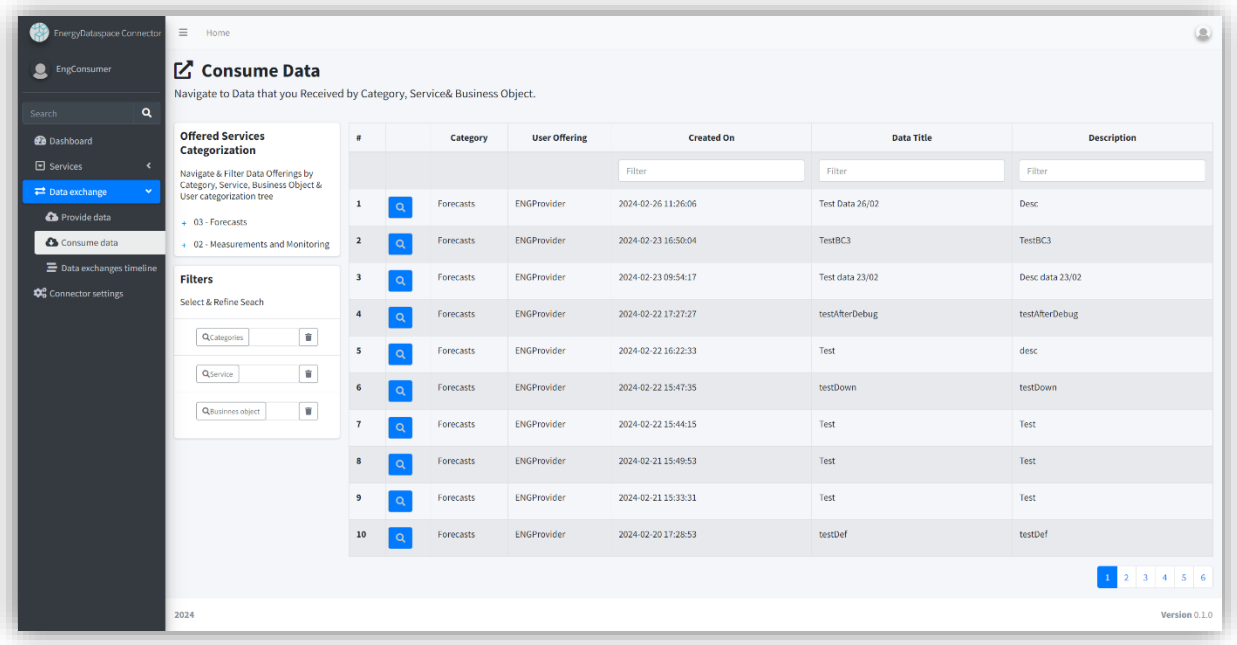

Figure 18: Consume Data

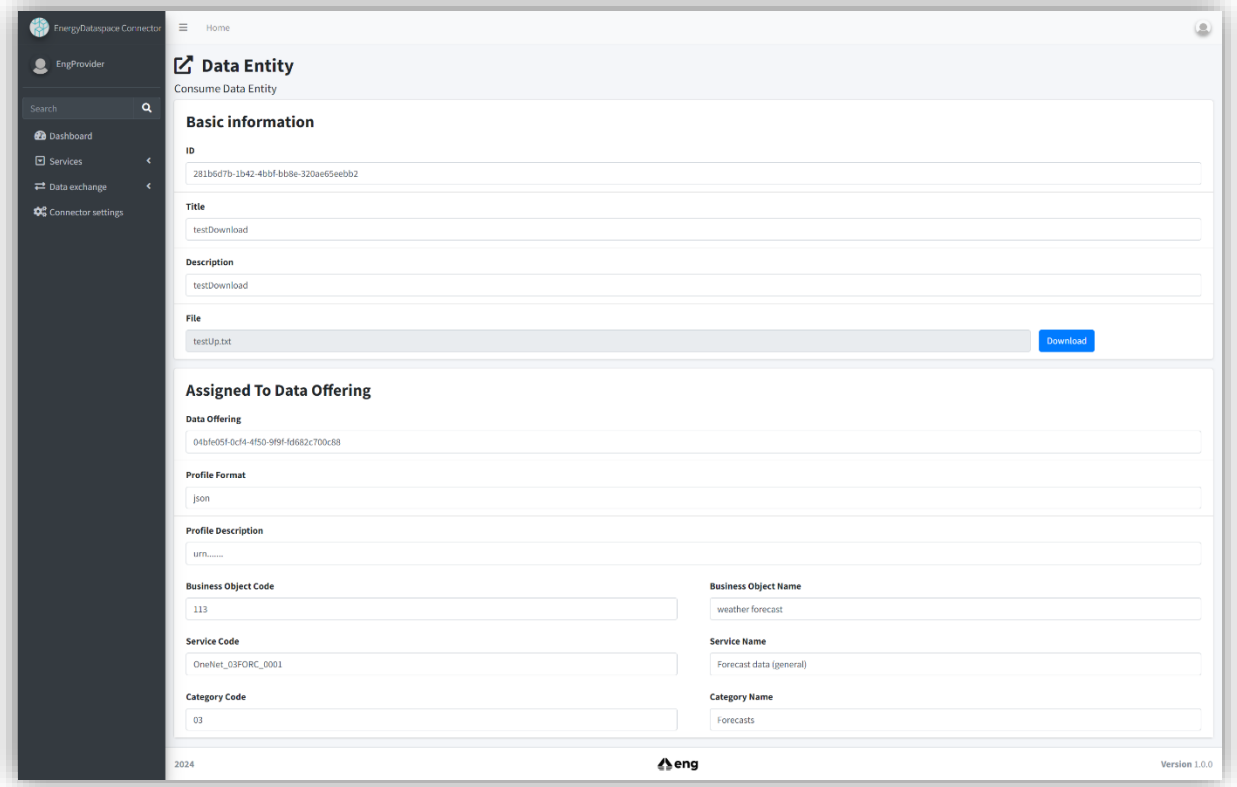

Figure 19: Data Entity Detail

3.2.10 Timeline

And finally, there is the data exchange list with the complete history of all data transactions. On the left we see the transaction in which the user acted as a provider, while on the right the one in which he acted as a consumer.

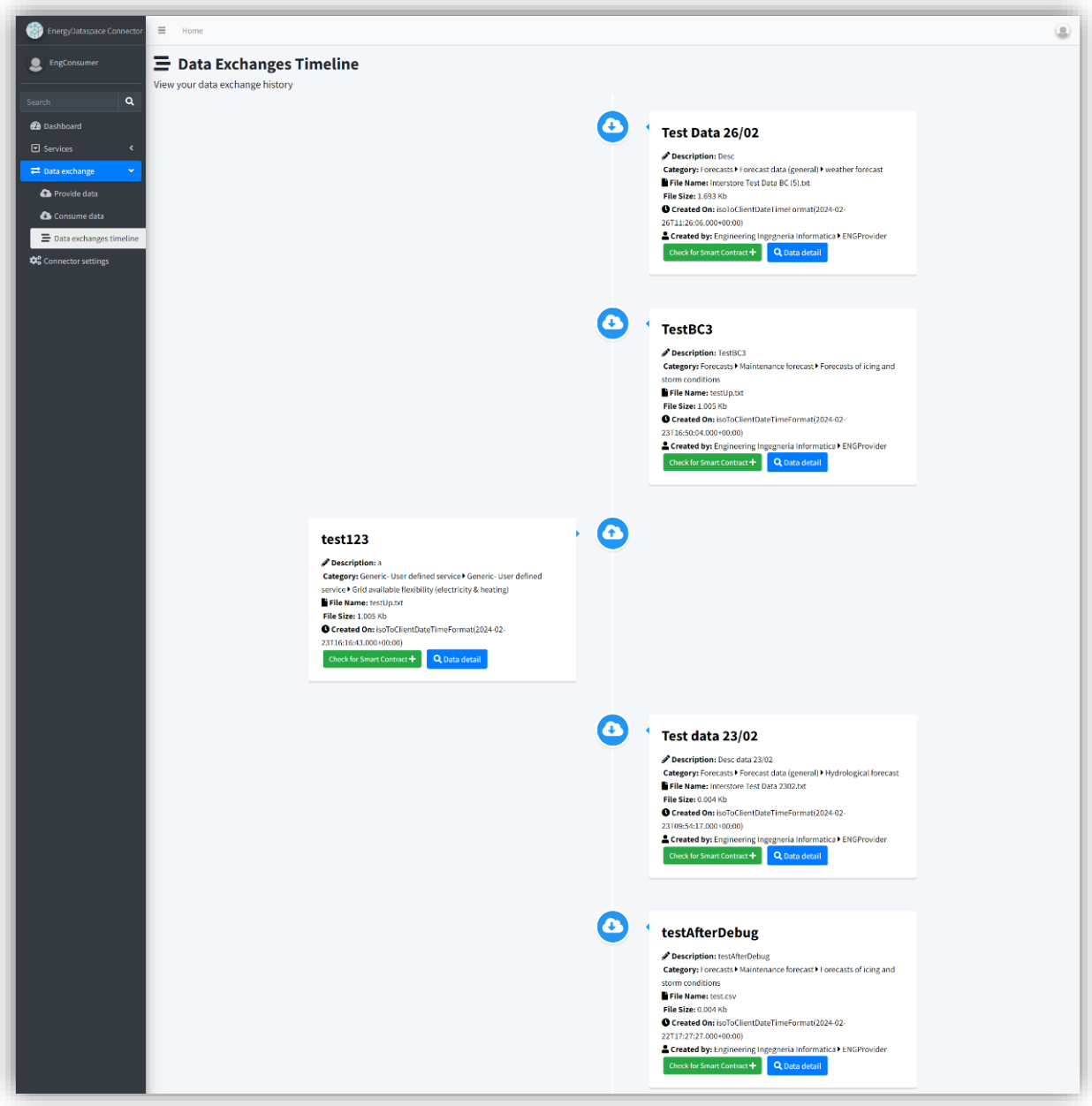

Figure 20: Timeline

3.2.11 Smart Contract

For each card of timeline interface, it's possible to see the main information with the entity's metadata and the possibility of checking for the presence of smart contracts.

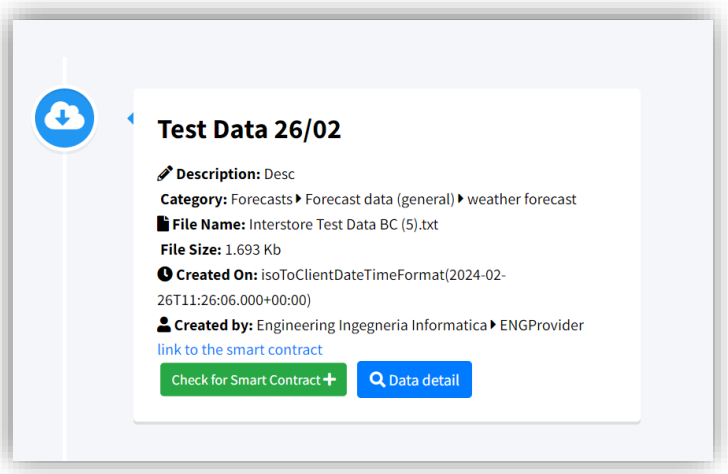

Figure 21: Timeline data exchanged card

The smart contract is created when the transaction between provider and consumer is completed with download of the file by the consumer.
A link allows to open a new browser tab and see the smart contract on blockchain explorer.

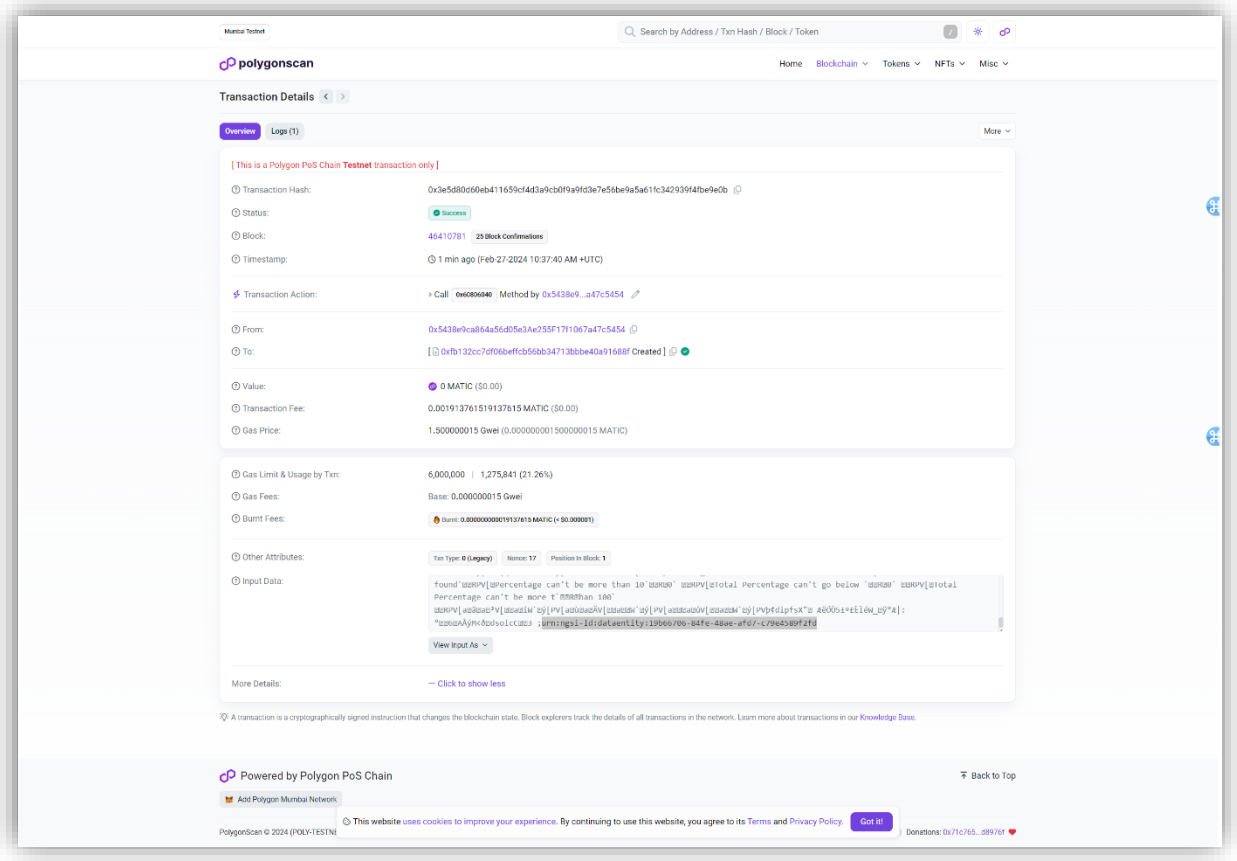

Figure 22: Smart Contract

4 Languages, Technologies and External Tools

Table 14: Languages, Technologies and External Tools

Component/Service	Languages	Technologies/Framework	External Tools
Connector UI	Typescript HTML5 CSS/SCSS	React	Nginx Docker
Middleware API	Java	Spring	Docker
Fiware Data App	Java	Spring	Docker
Database		MongoDB	Docker
Blockchain Notarization	Solidity	Truffle	Polygon Blockchain Nodes Docker
Execution Core Container	Java	Spring	Docker
Broker	C++	Fiware Orion	Docker

5 Deployment and availability

5.1 Deployment

The deployment process involves the use of Docker containers. The use of Docker guarantees not only an easy deployment process and total portability of the solution, but also a high level of scalability of the released applications.

The hardware and operating system prerequisites are:

1	A 64bit 2-core processor
2	8GB RAM Memory
3	50GB of disk space or more

The software prerequisites include:

1	Linux or Windows (preferably Server edition) Operative System (OS);
2	docker and docker-compose;

Energy Data Space Connector software and its components will be delivered utilizing the Docker containers functionalities. Firstly, the Docker platform has to be downloaded and installed accordingly to the OS of the server to host the deployment.

For the correct installation of docker and docker-compose, please refer to the official guides: https://docs.docker.com/get-docker/

Energy Data Space Installation on Docker

To proceed with the installation of Energy Dataspace Connector, the user must use the docker folder of the github repository that contains all the necessary configuration.

1. The first step is to clone this repository https://github.com/Horizont-Europe-Interstore/Energy-Data-Space-Connector in a specific folder *energy-data-space-connector*, by typing:

```
mkdir energy-data-space-connector
cd energy-data-space-connector
git clone https://github.com/Horizont-Europe-Interstore/Energy-Data-Space-Connector.git
```

2. There is the *docker-compose.yml* file located under the docker folder that contains all the configuration of the Energy Data Space Connector containers. Go to that file by typing the command:

```
cd energy-data-space-connector/docker
```

3. Start the containers with the below commands:

```
docker-compose up -d --build
```

4. To show logs use the command:

```
docker-compose logs -f
```

5. If no errors are seen, this means that Energy Data Space Connector was successfully deployed on your premisses.

Login & Connector Settings

The user interface is in a container that was installed on your premisses on the previous step. It can be accessed through the url:

```
http://localhost:8080/
```

1. You should see this interface, login using the username & password that you received from the Interstore Connector administrator.

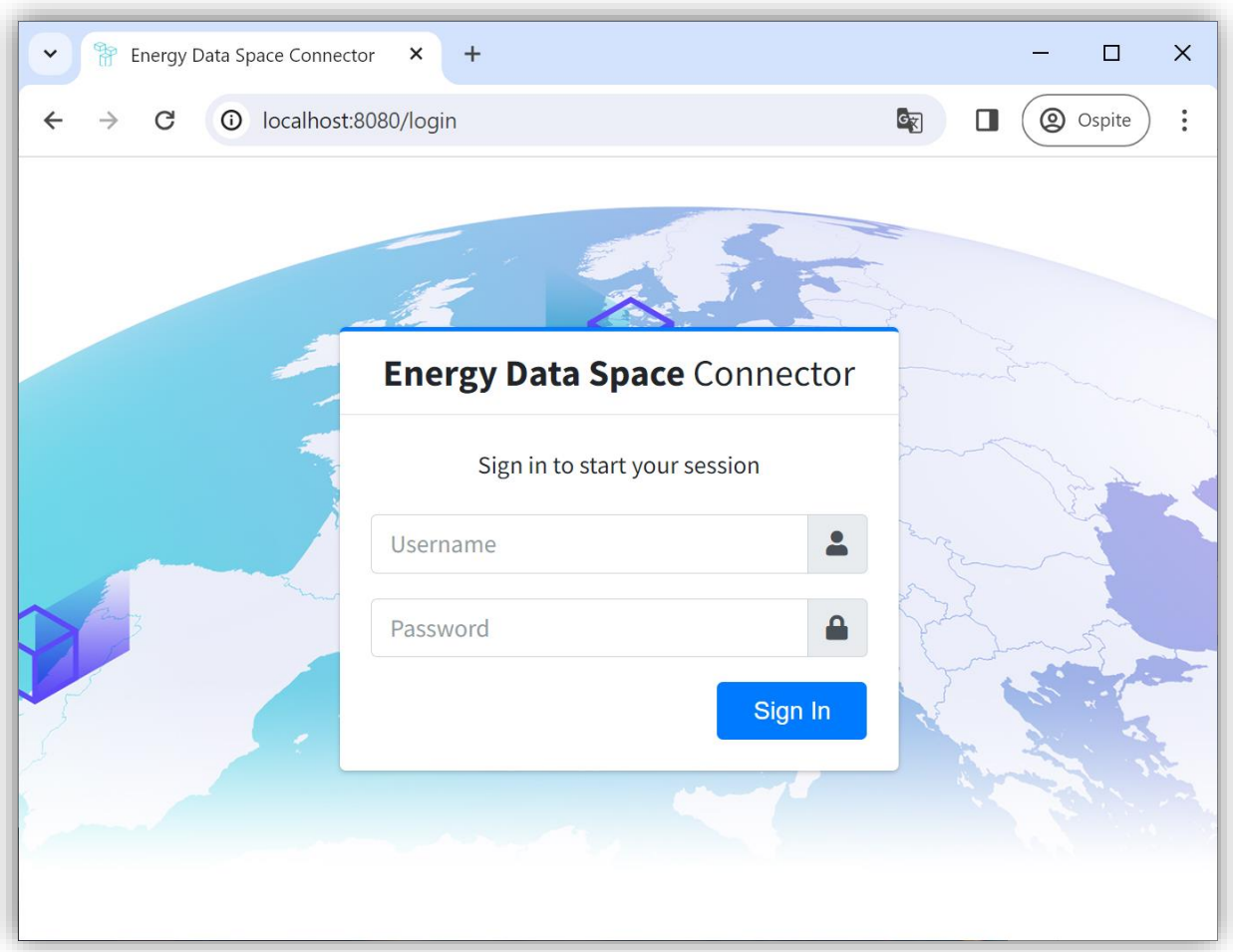

Figure 23: Login Page

2. Navigate to the connector settings by the sidebar menu & define the urls of your Local Api Url, Data App Url, Ecc Url.

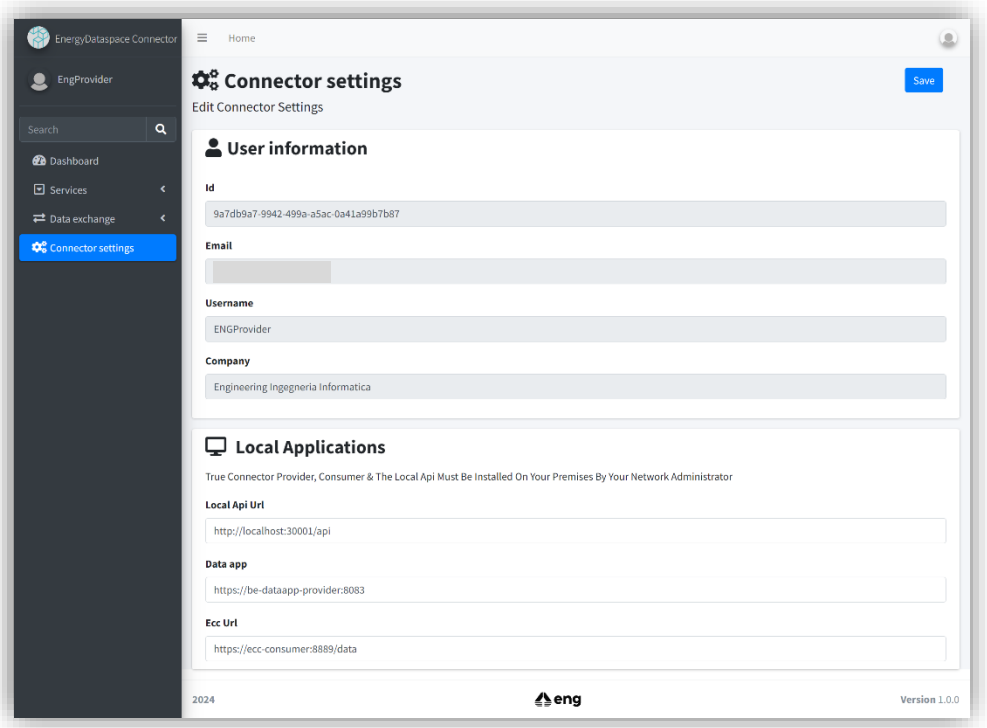

Figure 24: Connector Settings

Those 3 connector applications are running on the containers that you installed, so the urls must be configured accordingly as shown below.

Local Api Url

The url must be http://your_ip_where_the_containers_are_installed:30001/api

Data App Url

In the default testing configuration the given app is exposed to the URL https://be-dataapp-provider:8083 or https://be-dataapp-consumer:8084.

If you use an external Data app, the Data App must be publicly exposed with a static ip via https, before saved on the connection settings. This happens because Data App is served as an endpoint for peer to peer file transfer between you and other Data Space Connector users. So the url must be the https://your_static_url_that_points_to_dataapp_container.

Ecc Url

In the default testing configuration the ECC is exposed to the URL https://ecc-consumer:8889/data or https://ecc-provider:8889/data.

If you use an external ECC, the url must be publicly exposed with a static ip via https, before saved on the connection settings. This happens because Ecc Url is served as an endpoint for peer to peer file transfer between you and other Data Space Connector users. So the url must be the https://your_static_url_that_points_to_ecc_url_container

Environment Configuration

Inside the docker project folder, there is an *.env* environment configuration file. This file allow you to set all Back End configurations of the Data Space Connector.

There are many environment settings, the following table shows some settings.

Table 15: Environment Settings

Setting	Default value	Description
MONGO_HOST_IP	db-mongo	MongoDB host name or IP
MONGO_HOST_PORT	27017	MongoDB port
CONTEXT_BROKER_PROTOCOL	http	Context Broker protocol
CONTEXT_BROKER_IP	fiware-orion	Context Broker IP
CONTEXT_BROKER_PORT	1026	Context Broker port
NOTARIZATION_ENABLED	true	Allows you to enable or disable the blockchain notarization service. When disabled, no smart contract will be deployed at the end of the transaction.

5.2 Availability

The source code and the docker images necessaries for the deployment are available in the github repository and in public DockerHub of INTERSTORE project.

Github Repository

https://github.com/Horizont-Europe-Interstore/Data-Space-Connector

Docker Hub Repository

https://hub.docker.com/repositories/interstore

6 Conclusion

The work done at this stage provided the first prototype of the Interoperable Data Space Framework that enable the creation of an Energy Data Space for Interstore, integrating and deploying the Energy Data Space Connector.

The first release addresses a subset of requirements for integrating the Energy Data Space Connector within the demo sites and in particular: evolution of the TRUE Connector and OneNet Connector for Energy Data Space; adapted and improved Open GUI for Interstore pilots participants; blockchain-based notarization services for data transactions.

This first version of the Energy Data Space Connector will be adopted and integrated in several Interstore Pilots for testing the data governance enabled by the Energy Data Space approach.

The integration and deployment aspects will follow in T4.2 while the validation and collection of a first round of feedback will be implemented in WP5.

After the first validation of the Interoperable Data Space Framework, a new version including new services as well an evolution of the Data Space Middleware will be provide on March 2025.

7 REFERENCES

[1] https://international-data-spaces-association.github.io/DataspaceConnector/Introduction
[2] https://fiware-orion.readthedocs.io/en/master/
[3] https://ethereum.org/
[4] https://chain.link/education-hub/what-is-layer-2
[5] https://www.ledger.com/academy/topics/blockchain/what-are-ethereum-layer-2-blockchains-and-how-do-they-work
[6] https://www.openapis.org/

8 LIST OF TABLES

9 LIST OF FIGURES