Ref. Ares(2025)2618765 - 01/04/2025

Federated decentralized trusted dAta Marketplace for Embedded finance

D4.5 - Pricing, Trading and Monetization Techniques II

Title	D4.5 - Pricing, Trading and Monetization Techniques II
Revision Number	1.0
Task reference	T4.2 T4.3 T4.4
Lead Beneficiary	FTS
Responsible	Geert Machtelinckx
Partners	ENG, FTS, INNOV, JOT, JRC, TRB, UBI, UIA
Deliverable Type	DEM
Dissemination Level	PU
Due Date	2025-03-31 [Month 27]
Delivered Date	2025-04-01
Internal Reviewers	IBM ECO
Quality Assurance	GFT
Acceptance	Coordinator Accepted
Project Title	FAME - Federated decentralized trusted dAta Marketplace for Embedded finance
Grant Agreement No.	101092639
EC Project Officer	Stefano Bertolo
Programme	HORIZON-CL4-2022-DATA-01-04

This project has received funding from the European Union's Horizon research and innovation programme under Grant Agreement no 101092639

©FAME Consortium

Revision History

Version	Date	Partners	Description
0.1	2025-02-01	FTS	Table of Contents
0.2	2025-03-05	FTS ENG INNOV JOT JRC TRB UBI UIA	Contribution from WP4 partners
0.3	2025-03-19	FTS	Final Draft
0.8	2025-03-24	ECO IBM	Version for peer review
0.9	2025-03-31	GFT	Quality Assurance
1.0	2025-04-01	FTS	Version for Submission

Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union.
Neither the European Union nor the granting authority can be held responsible for them.

Definitions

Acronyms	Definition
AAI	authentication authorization infrastructure
AI	Artificial Intelligence
AID	Asset Identifier
AP	Asset Pricing
API	Application Programming Interface
CD	Continuous Development
DB	Data Base
ERC	Ethereum Request for Comments
FAME	Federated decentralized trusted dAta Marketplace for Embedded finance
FDAC	Federated Data Assets Catalogue
GOV	Operational Governance
HTTP	HyperText Transfer Protocol
ID	Identity
JSON	JavaScript Object Notation
JWT	JSON Web Token
MVP	Minimum Viable Product|Platform
NLP	Natural language processing
PAT	Pricing Advisory Tool
PTM	Pricing, Trading and Monetisation
REST	Representational State Transfer
RPC	Remote Procedure Call
SAT	Similarity Advisory Tool
SQL	Structured Query Language
SSE	Semantic Search Engine
TID	Trade Account Identifier
UI	User Interface
URL	Uniform Resource Locator

Executive Summary

In the context of FAME, “Pricing, Trading, and Monetization Techniques” are fundamental to enable the fundamental feature of a federated Marketplace to provide a way to exchange assets between producers and consumers.

This deliverable is the second iteration of the document, showing the tangible result of the combined efforts done under “Pricing, Trading, and Monetization Techniques”. It describes the artifacts created in tasks T4.2 “Accounting, Trading, Pricing and Monetization Schemes”, T4.3 “Smart Contracts for Programmable Trading and Monetization”, and T4.4 “Semantic Search for Trading and Valuation of Data Assets”, in a form of an integrated demonstrator that is documented in this document.

The three tasks described focus on separate modules that are not tightly coupled.

Task T4.2 is both about a Price Advisory Tool and a Similarity Analysis Tool, helping data asset owners set the right price for their digital asset. T4.3 delivers smart contract tooling that enables the creation of a digital asset offering (represented as a token on the platform), as well as the execution of a trade on a digital asset between consumers and sellers. T4.4 allows users to search for digital assets, based on the FDAC’s asset metadata.

The document outlines the logic and technical specifications of the first release of the software modules provided by these tasks.

Key Insights:

- The prototypes are as such not closely interrelated, but all contribute to an overall service offering that will be made available through a ‘Dashboard’, developed in WP2.
- All prototypes integrate closely with other tasks and deliverables such as the Federated Catalogue of Data Assets, the Blockchain-based Data Provenance Infrastructure and the Operational Governance tool, developed in WP4.
- All also relate to the work on the Platform Architecture, the Data Marketplace Platform Integration, the Federated AAI Infrastructure, the Unified Security Policy Management, and the Data Provenance Infrastructure, developed inWP3.

In particular, the document provides information related to:

Modules Overview: Describes the different modules’ role and contribution to the FAME federation marketplace.

Components Specification: Details the technical specifications of each module’s components.

Modules Demonstration: Offers instructions on setting up the prototype and provides a visual demonstration of the workflows.

The document also provides an Installation Guide for the Smart Contract Main Infrastructure Server and emphasizes the MVP nature of the current release, with future enhancements planned for the FAME project’s second phase.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Objective of the Deliverable

One of the main objectives of the FAME project is to design and implement pricing, trading, and monetization mechanisms for a federated data marketplace infrastructure. These mechanisms are integral elements of the FAME Platform. In this context, this document is mainly a *factsheet* describing the software prototype released as deliverable D4.5 “Pricing, Trading, and Monetization Techniques II”, which is an evolved version of the previously published Deliverable D4.2 titled “Pricing, Trading and Monetization Techniques I”. This second release includes new features and shows tight integration with the other Platform modules, ex. a description of the “Data Gateway”.

In its current form, the D4.5 prototype has been tested and demonstrated in the scope of the public release of the FAME Platform.

The output of T4.2 is a Pricing Advisory Tool, known as the PAT, a data-driven pricing advisory mechanism leveraging issuer-provided data, stakeholder survey responses, and historical pricing realization analytics. It is a tool to guide producers in setting a relevant price for digital asset while considering its intrinsic value, leaving the final price decision solely to the producer. PAT contains back-end functionalities for price-advice calculation, and it includes the Similarity Analysis Tool, known as SAT, which purpose is to search for similar assets and for which historical sales have been executed. At this stage, we use PAT as a tool to guide business offering creators in setting prices by calculating the value of the asset based on a cost-based approach and the prices of similar assets. We use a demand-based approach (represented by the number of transactions of similar assets) for price suggestion calculation.

The deliverable of T4.3 (smart contracts for Trade and Monetization) contains an integrated demonstrator and includes the back-end functionalities available to set up the smart contracts that allows the creation of offers and trading initiation, i.e. exchanging the data asset with a payment token in the context of different payment schemes (over the counter, pay as you go, pay as you use, subscription). This back-end feature is being used in a user front-end dashboard.

The semantic search component (created through task T4.4) focuses on providing an intuitive and effective search functionality in the FAME Platform. It processes user queries to retrieve relevant FDAC data assets, leveraging semantic analysis to ensure that the results align with the user’s search intent. This component supports the wider aim of making it easier to find data assets, setting the stage for later combining it with the PAT to create a full-fledged asset trading and pricing instrumentation.

The Token Gateway System, introduced as part of this deliverable, plays a critical role in bridging the gap between tokenized data access rights and actual data consumption in the FAME Platform. Designed to be implemented and deployed by each Data Provider, it is responsible for validating Data Access Tokens (DATs) and enforcing access policies in real time before any data is served. By integrating with the FAME Trading & Monetization System, the Token Gateway evaluates token ownership, expiration, and usage conditions. This system ensures that data access is securely gated, policy-compliant, and verifiable, empowering Data Providers to maintain full control over their assets while relying on federated, contract-based logic for enforcement.

1.2. Insights from Other Tasks and Deliverables

Deliverable D4.5 is the result of activities performed in the scope of three tasks: T4.2 “Accounting, Trading, Pricing and Monetization Schemes”, T4.3 “Smart Contracts for Programmable Trading and Monetization”, and T4.4 “Semantic Search for Trading and Valuation of Data Assets”. It is built on

top of foundational tasks such as T3.3 “Federated Catalogue of Data Assets”, T4.1 “Decentralized Data Provenance and Traceability”, T3.1 “Federated AAI Infrastructure”, and T4.5 “Business and Operational Models for the FAME Marketplace”.

The tasks are executed following an architectural design that was defined in T2.2 “Platform Architecture and Technical Specifications” and has been integrated in T2.3 “Data Marketplace Platform Integration”.

These interdependencies of relationships are depicted in **Error! Reference source not found.**.

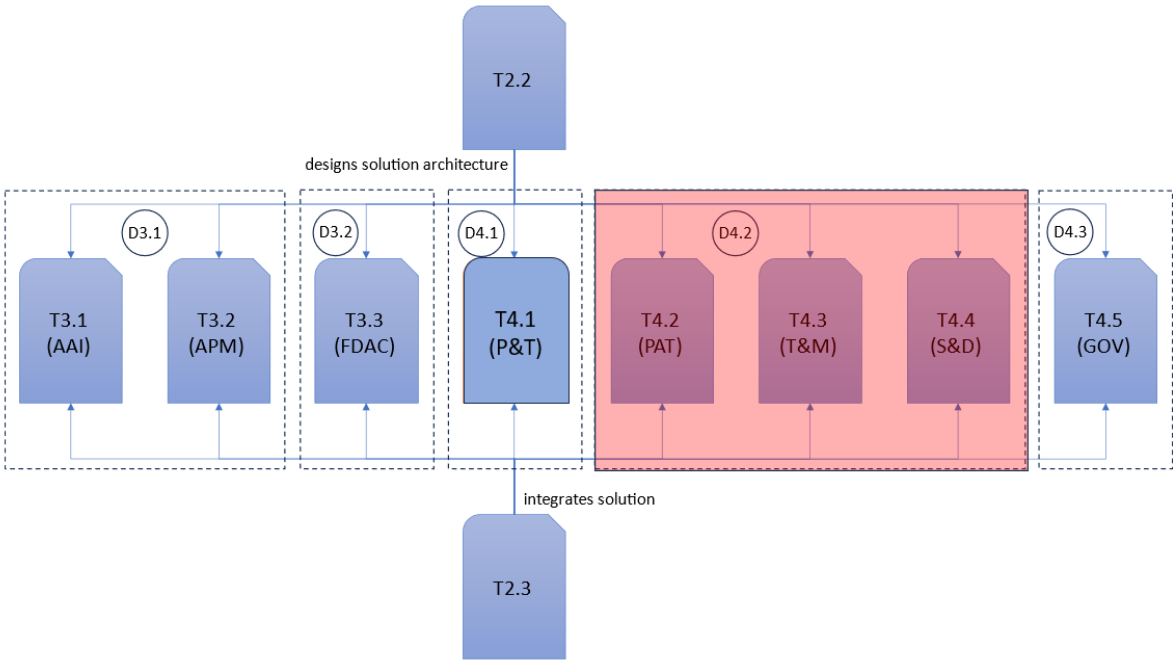

Figure 1 – Tasks / deliverable relationship

1.3. Structure

The remaining of this document consists of four sections:

1. Module Overview: This sets the context (with reference to the FAME Solution Architecture from deliverable D2.2), gives a general description of the module from its functional perspective, identifies the software Components, and explains their relationship with other modules in the FAME Platform.
2. Components specification: This provides the complete technical specifications of each of the module’s Components. These include the baseline technologies, interfaces and data structures that are used internally (for data persistence) and externally (for interoperability).
3. Module Demonstration: This gives instructions on how to set up the prototype in a suitable environment and leads the reader through visual demonstrations (represented by screenshots) within a partially integrated FAME Platform.
4. This section provides a recap of the main achievements and an outlook for future activities.

2. Modules Overview

The Pricing, Trading, and Monetization (PTM) module in the FAME Platform is responsible for trading activities. To represent and describe it, in this document we leverage the C4 model for visualizing software architecture. The C4 model divides the system's architecture into four hierarchical levels, as described in the following diagram and description:

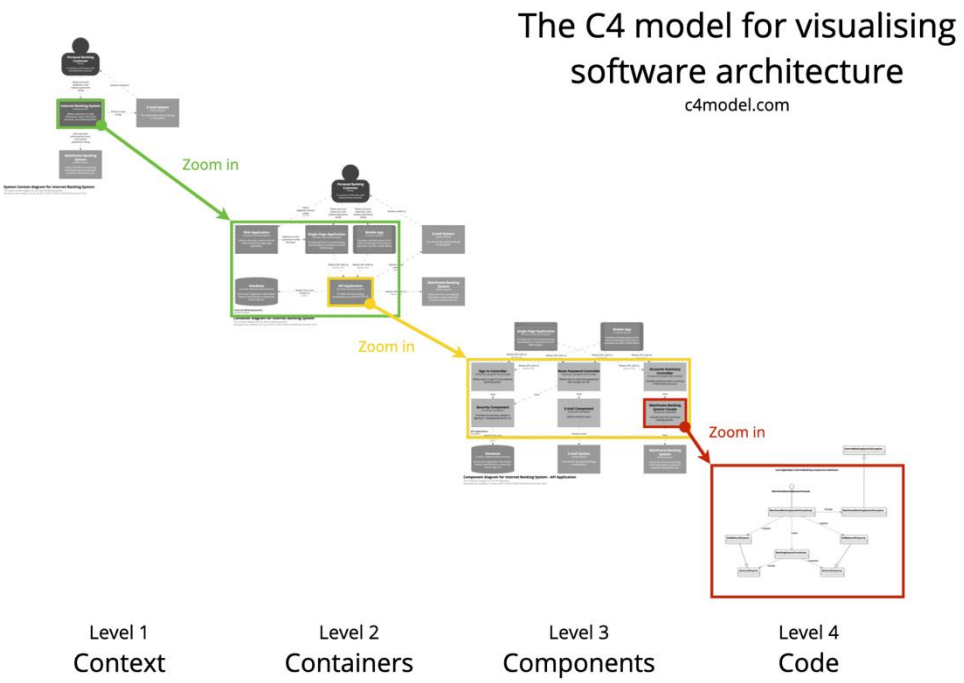

Figure 2 – The C4 Model for Visualizing Software Architecture

- **Context Diagram (Level 1):** Provides an overarching view of the system within its operational environment. It identifies key external actors (such as users, dashboards, and external services) and outlines their interactions with the system.
- **Container Diagram (Level 2):** Decomposes the system into its major execution environments or containers (for example, web servers, backend services, and smart contracts). This level clarifies the responsibilities of each container and how it communicates via defined interfaces.
- **Component Diagram (Level 3):** Breaks down each container into individual components. It details the internal structure, the roles of the components and their interrelationships, thereby revealing the system's modular design.
- **Code Diagram (Level 4):** Offers a granular view of the implementation by detailing the classes, methods, and libraries. This level is intended for developers who need to understand or modify the underlying codebase. We do not cover such details in this document.

The subsequent sections in this chapter are organized according to the C4 levels, beginning with the Context Diagram to present the high-level interactions and then delving deeper into the Containers and Components to provide a comprehensive understanding of the system's design.

In **Error! Reference source not found.** the modules described in this deliverable D4.5 are indicated in red. They contribute to the overall architecture (full picture) and interact with other modules. The overall architecture and the role of each module is described in architecture delivery D2.6.

According to the C4 architecture model, the Pricing, Trading, and Monetization (PTM) module should be represented as three distinct systems that interacts with each other: "Trading & Monetization" (T&M), "Asset Pricing" (AP) and "Semantic Search Engine" (SSE). We have added also the "Token Gateway" to the architectural overview, a module that has been added to the architecture recently.

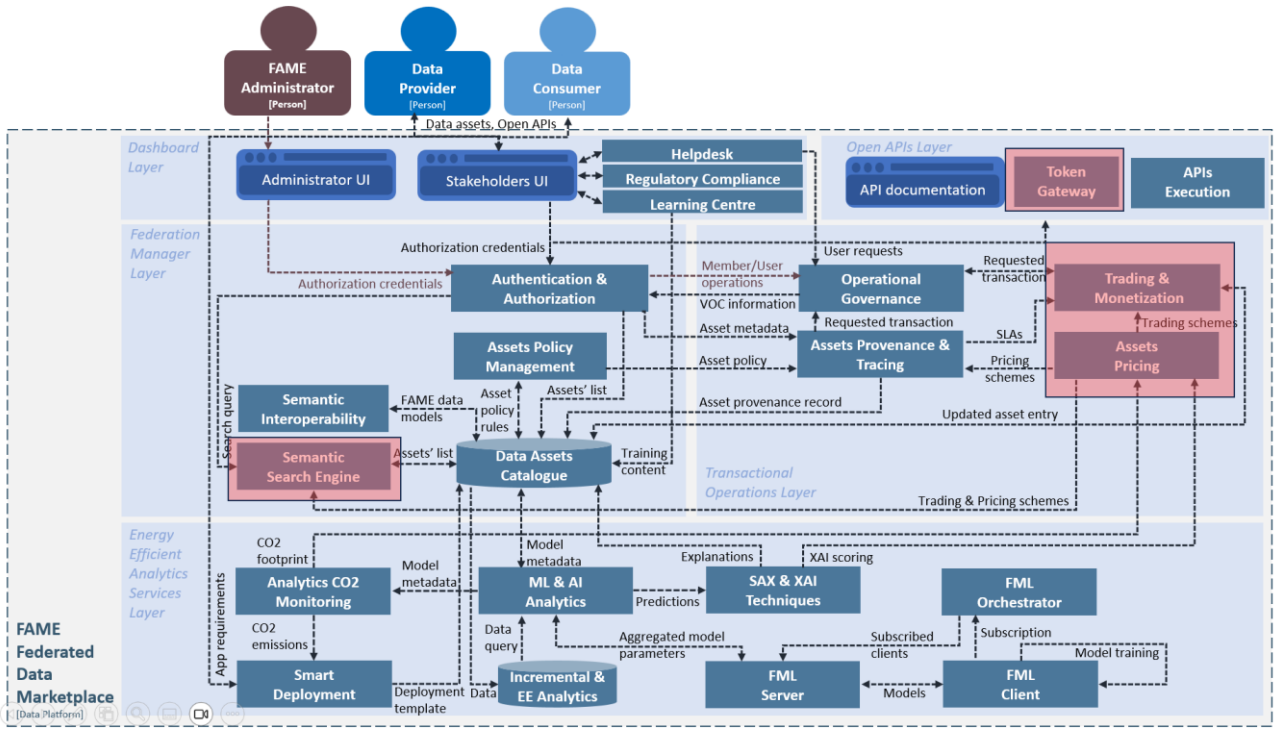

Figure 3 – Modules Overview

2.1. Trading and Monetization (T&M)

In this chapter, we detail the architecture of the Trading and Monetization System (T&M) using the C4 model.

2.1.1. System Context (Level 1) of T&M

The high-level overview of the T&M system, its users, and their interactions are depicted below:

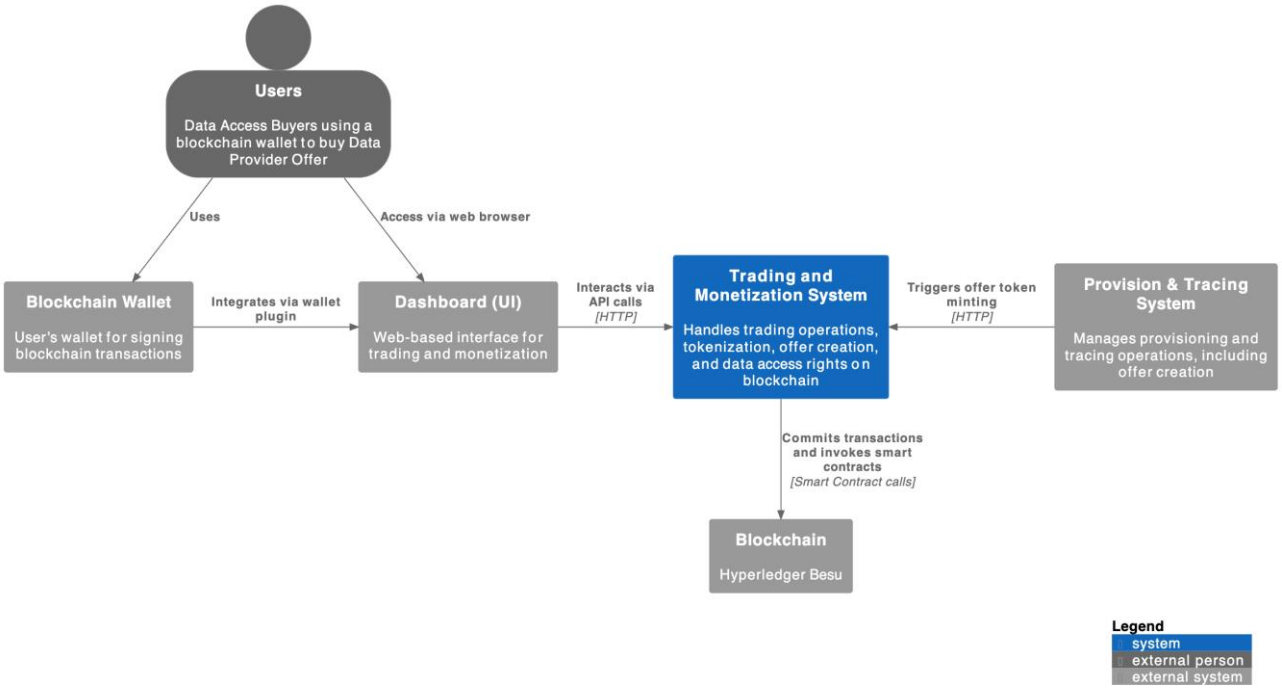

Figure 4 – Context Diagram for Federated Data Asset Trading Platform

Context Level 1 in Figure 4 shows the high-level ecosystem where the **Trading and Monetization System** serves as the central hub for managing trading operations, tokenization, offer creation, and data access rights in blockchain. The diagram emphasizes the interactions between the internal components and several external systems, which together constitute a trading environment.

- **Users:** Data Access Buyers who utilize a blockchain wallet to purchase data provider offers. They interact with the system primarily via a web browser and using their blockchain wallet.
- **Dashboard (UI):** A web-based interface that enables users to navigate the trading and monetization functionalities. It integrates with the blockchain wallet (via a wallet plugin) and communicates with the T&M System through API calls.
- **Blockchain Wallet:** A critical external system used by the users to sign blockchain transactions. It provides the necessary cryptographic functionality to authorize and secure the interactions with the system.
- **Provision & Tracing System:** This system manages provisioning and tracing operations, including the creation of offers. It triggers the minting of offer tokens by communicating with the T&M System over HTTP.
- **Blockchain System** - The underlying blockchain platform that records all transactions and smart contract invocations. The T&M System commits transactions to the blockchain (implementation using Hyperledger Besu technology), ensuring decentralized and immutable ledger operations.

The diagram outlines how these components interact: users access the dashboard, which integrates with their blockchain wallet; the dashboard then communicates with the T&M System via HTTP API calls. Additionally, the Provisioning & Tracing System triggers token minting within the system, while all blockchain-related transactions and smart contract calls are committed to the Hyperledger Besu component/service.

2.1.2. System Containers (Level 2) of T&M

The following diagram represents the container level of the C4 model, zooming into the "T&M System" containers of the overall platform. This focuses on the specific responsibilities and functionalities of that system, breaking it down into multiple interacting containers.

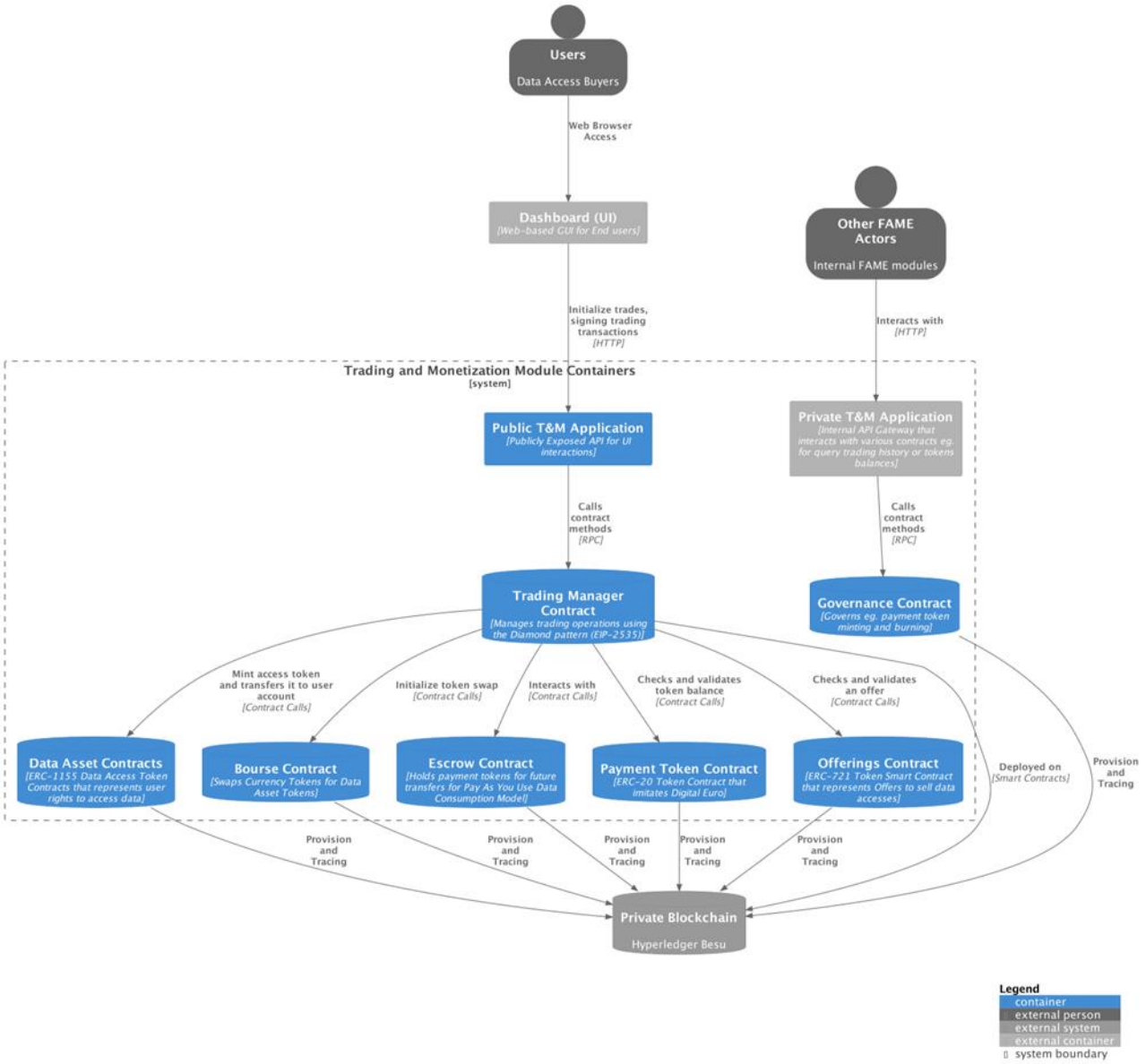

Figure 5 – T&M System C4 Containers Level 2 Diagram

- **Other FAME modules**: Represents internal modules in the FAME platform that interact with the T&M system. These are other backend modules that operate within the FAME ecosystem, utilizing the T&M API for seamless interaction with the blockchain and tokenized assets.
- **Users (Data Access Buyers)**: End users who purchase access to data through the system. They interact with the system primarily through the dashboard, browsing available data offerings, making purchases using FDE tokens, and managing their digital assets.

- **Private Blockchain**: A permissioned blockchain module used for managing smart contracts related to trading and payments. It ensures secure and immutable storage of transactions related to data asset ownership, payments, and trading operations. All token-related activities, including minting, burning, and exchanging tokens, are recorded in this blockchain for transparency and traceability.

- **Dashboard** (Web UI, External Container): A graphical interface used by the end users (buyers) to interact with the T&M system. The dashboard provides a user-friendly experience for browsing data assets, executing transactions, and tracking asset ownership. It interacts with the Public T&M Application (see below) to fetch available offerings, execute purchases, and manage tokenized assets.

- **Private T&M Application** (Private API): An internal API that allows the FAME systems to interact with the trading platform. This API is not publicly exposed and is only accessible by internal FAME backend modules. It acts as a bridge between its internal systems and smart contracts, facilitating transactions and governance-related operations in a secure environment.

- **Public T&M Application** (Public API): A publicly exposed API for external users and services. This API enables external actors to interact with the T&M system, allowing them to access available data offerings, execute trades, and manage their tokenized assets. It is designed with open standards to ensure compatibility and accessibility while maintaining security and compliance.

The diagram above (Figure 5) also presents, as separate containers, all smart contracts that are part of the T&M system. These contracts handle various aspects of token transactions and data asset management.

- **Trading Manager Contract**: Manages trading operations, and orchestrates core trading workflows by validating token balances via the Payment Token Contract, verifying offers through the Offerings Contract, initiating token swaps with the Bourse Contract, interacting with the Escrow Contract for secure fund handling, and minting data access tokens via the Data Asset Tokens Contract.

- **Payment Token Contract** (ERC-20): This smart contract is responsible for handling the minting and burning of FDE tokens, which serve as the primary currency for transactions within the system. It interacts with the Governance Contract to ensure that token issuance and destruction are managed according to predefined rules. This contract also facilitates the transfer of tokens between users as they engage in data purchases and other economic activities in the platform.

- **Data Access Subscription Contract** (ERC-1155): This contract represents a data access model where users purchase access to a data asset for a fixed period, typically on a subscription basis. Once a user acquires a subscription-based data access token, they can access the corresponding data asset as long as their subscription remains active.

- **Data Access Pay-As-You-Go (PAYG) Contract**: This contract represents a business model where users pay for data access based on actual usage rather than a fixed subscription. Users acquire data access tokens that grant them a limited amount of usage rights, such as a specific number of API calls, downloads, or access durations. Once the allocated usage limit is reached, the token expires, requiring the user to purchase additional access. This model is suitable for users who do not need continuous access but prefer to pay for data only when needed.

- **Data Access Pay-As-You-Use (PAYU) Contract**: This contract is designed for a deferred payment model where users can access data first and pay later based on actual consumption. The contract interacts with the escrow system, holding the payment tokens in reserve until the data provider confirms the user's usage. If the agreed-upon conditions are met, the funds are released to the provider. If the user does not fully utilize the data or disputes arise, the contract can enforce rules for partial refunds or alternative settlements. This model is particularly useful for enterprise clients or use cases where real-time billing based on exact consumption is required.

- **Offerings Contract**: This smart contract functions as a ledger for trade offers, allowing data providers to list their data assets for sale and users to browse and purchase them. The contract keeps track of active offerings, manages pricing and access conditions, and ensures that transactions adhere to the predefined business models, such as pay-per-use or subscription-based access.

- **Bourse (Trading) Contract**: This contract facilitates the core trading mechanism, enabling the exchange of currency tokens (FDE) for Data Access Tokens. It supports various trading models, including direct sales and escrow-based transactions. The contract ensures that buyers receive the correct data asset tokens upon successful payment and that the funds are correctly allocated to data providers or escrow accounts as needed.

- **Governance Contract**: The Governance Contract plays a crucial role in overseeing the minting and burning of tokens and enforcing platform-wide policies. It interacts with multiple other contracts, including the Payment Token Contract, Escrow Contract, and Trading Contract, to regulate token supply and implement governance decisions. This contract may also incorporate decentralized governance mechanisms, allowing stakeholders to vote on protocol updates and policy changes.

- **Escrow Contract**: This contract provides a secure mechanism for temporarily holding payment tokens before finalizing transactions. It is particularly useful in pay-per-use models, where funds need to be held until certain conditions are met, such as confirming that the buyer has accessed the purchased data. Once the conditions are satisfied, the Escrow Contract releases the funds to the data provider. If the conditions are not met, the contract may refund the buyer or reallocate the tokens according to the predefined business logic.

Interactions between containers

The T&M System is a seamlessly interconnected ecosystem of smart contracts, APIs, and a user-facing dashboard, designed to provide an automated and efficient marketplace for data asset transactions. The entire process begins when a data access buyer accesses the system through the Dashboard (Web UI). This web-based interface offers a streamlined experience for browsing data offerings, initiating purchases, and managing acquired tokens.

When a user decides to purchase access to a data asset, the Dashboard communicates with the Public T&M Application (Public API) over standard HTTP protocols. Acting as an external gateway, the public API allows users and third-party services to interact with the system. However, it does not directly execute blockchain transactions or interact with smart contracts. Instead, it prepares an unsigned transaction for purchasing data access tokens. This unsigned transaction is then sent back to the Dashboard, where the user must sign it using their blockchain wallet. Once signed, the Dashboard broadcasts the transaction to the blockchain network, ensuring that the process remains decentralized, and user-controlled.

The internal FAME modules, collectively referred to as Other FAME Actors, interface with the system through the Private T&M Application (Private API). Unlike the Public API, this Private API is exclusively accessible by the internal FAME backend services and enables authorized modules to execute governance-related operations and register new data offerings. When a data provider creates a new data offering, the Private API records it in the Offerings Contract, making it available for purchase through the Public API.

A key orchestrator within this ecosystem is the Trading Manager Contract. Invoked via Remote Procedure Calls (RPC) from the Public T&M Application, it coordinates various trading operations. It validates token balances by interacting with the Payment Token Contract, checks for offer validity with the Offerings Contract, initiates token swaps through the Bourse Contract, interacts with the Escrow Contract for secure transactions, and mints data access tokens by calling the Data Asset Tokens Contract. All of these interactions occur in the Hyperledger Besu blockchain, which maintains a secure and immutable ledger.

The financial backbone of the system relies on the Payment Token Contract (ERC-20), which governs the minting, burning, and transfer of FDE tokens, the currency used within the platform. However, this contract does not function in isolation; it operates under the oversight of the Governance Contract, which ensures that token-related activities comply with platform rules. Any request to mint or burn FDE tokens must be validated by the Governance Contract, which then executes the necessary blockchain transactions to adjust token supply accordingly.

When a user completes a signed purchase transaction, the system selects the appropriate Data Access Contract based on the business model chosen by the user. The Data Access Subscription Contract grants access for a fixed period, ensuring that the user retains the right to access the data as long as their subscription remains valid. The Pay-As-You-Go (PAYG) Contract allows users to consume access incrementally, deducting usage rights progressively based on actual consumption. The Pay-As-You-Use (PAYU) Contract, on the other hand, introduces an additional layer of financial security by holding the payment tokens in escrow until the agreed consumption conditions are met.

All transactions, including the conversion of FDE tokens into data access tokens, take place within the Bourse (Trading) Contract. This contract acts as the exchange where users can swap FDE tokens for access rights to data assets. It ensures that buyers receive their purchased data access tokens while data providers are compensated fairly.

To enhance security and enforce compliance, the Escrow Contract plays a crucial role, particularly in PAYU transactions. It safeguards payment tokens by holding funds until the data provider confirms that the user has utilized the purchased data access. If conditions are met, the funds are released to the provider. If a dispute arises or the usage conditions are not fulfilled, the escrow mechanism ensures that the funds are handled according to predefined business logic, preventing unauthorized transfers and protecting both users and data providers.

2.1.3. Components (Level 3) of T&M

At the Component Level 3 of the C4 model, the objective is to provide a deeper, more technical breakdown of each container by decomposing them into their constituent components. The following descriptions present detailed technical insights into the individual containers.

2.1.3.1. Private Backend Server (Private API)

The Trading & Monetization system consists of two backend services: the **Private API** and the **Public API**. These services handle different functionalities of the platform, ensuring that internal FAME modules and external users can interact securely with smart contracts while maintaining clear access boundaries.

The Private API is designed exclusively for **internal use by other FAME modules**. It facilitates backend-to-backend communication, ensuring that only authorized services can perform critical blockchain operations. This API is not accessible to the public and implements strict access control policies to prevent unauthorized interactions with smart contracts.

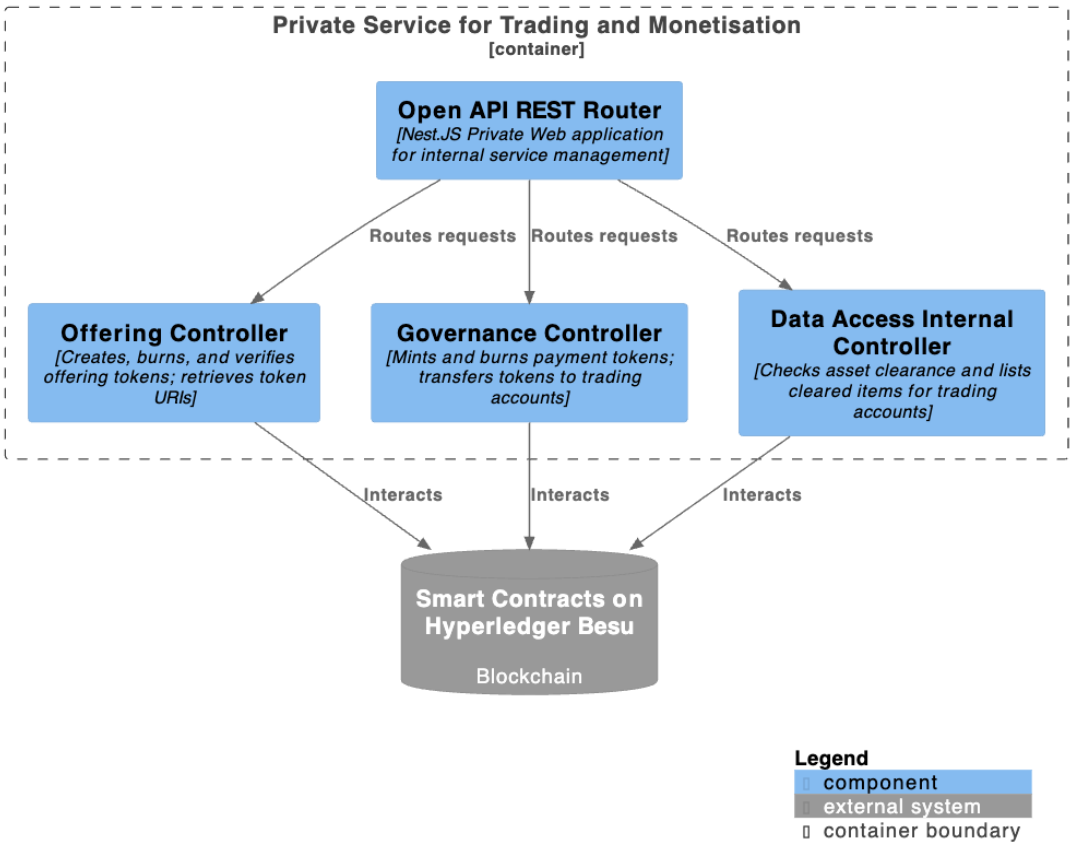

Figure 6 – Private App in Trading & Monetization Level 3 Component Diagram

The **Private Service for the T&M** container is a critical internal component that manages the core operations required for the secure administration of offerings and payment tokens. This container is purposefully isolated from public access, ensuring that only authenticated internal services and operators can perform sensitive actions. The architecture leverages a modular design that segregates functionality across several controllers, each responsible for a specific aspect of the system's operations, with all components interfacing directly with the Hyperledger Besu blockchain system. The container consists of the following main components:

1. **Open API REST Router (Nest.JS Component)**
   This component acts as the primary entry point for internal API requests. Implemented using the Nest.JS framework, it encapsulates the routing logic required to dispatch client requests to the appropriate controllers. By abstracting the routing mechanism, it enables a decoupled design where clients need not be aware of the underlying controller implementations, thus simplifying client interactions and enforcing consistent validation and access control.
2. **Offering Controller**
   The Offering Controller manages the lifecycle of offering tokens. It is responsible for creating (minting) new tokens on the blockchain by processing requests that include the necessary offering details – such as asset ID, offering ID, resource URI, beneficiary information, pricing, duration, and other operational parameters. The token creation process is secured by an operator's private key signature, ensuring that the minting operations are

both controlled and verifiable. In addition, this controller facilitates the burning (removal) of tokens, allowing obsolete or invalid tokens to be securely revoked.

3. **Governance Controller**
   The Governance Controller handles financial operations central to the system's internal economic management. It is tasked with both minting and burning payment tokens, as well as transferring these tokens to the designated trading accounts. This controller ensures that all payment-related operations are executed in a secure, auditable manner on the blockchain, thereby upholding the system's financial integrity and transparency.
4. **Data Access Internal Controller**
   This controller focuses on internal asset management and verification. It provides endpoints for checking whether a specific asset is owned by given trading accounts and for retrieving lists of cleared items associated with those accounts. Such functionality is critical for compliance, internal audits, and ensuring that access rights are correctly enforced across the trading ecosystem.

A significant aspect of the container's design is its integration with the blockchain, represented by Smart Contracts on Hyperledger Besu. Every controller within the container directly interacts with the blockchain system, ensuring that all operations – whether they involve token minting, burning, or asset clearance verification – are executed in an immutable and verifiable manner. This tight coupling with the blockchain not only enhances security but also provides an auditable trail of all internal operations, which is indispensable for regulatory compliance and trust.
Security within the container is enforced through rigorous access controls and the use of secure communication channels. As an internal service, it is shielded from public exposure, reducing the attack surface considerably. Moreover, the blockchain integration mandates that all transactions are cryptographically signed, ensuring that no unauthorized or accidental modifications can occur. This layered approach to security supports both the operational stability of the system and the integrity of the data processed within it.

2.1.3.2. Public Backend Server (Public API)

The Public API is designed for **external users and third-party integrations**, providing a controlled way for them to interact with the Trading & Monetization system. Unlike the Private API, this service is publicly accessible but includes **authentication, transaction signing, and rate limiting** to ensure security.

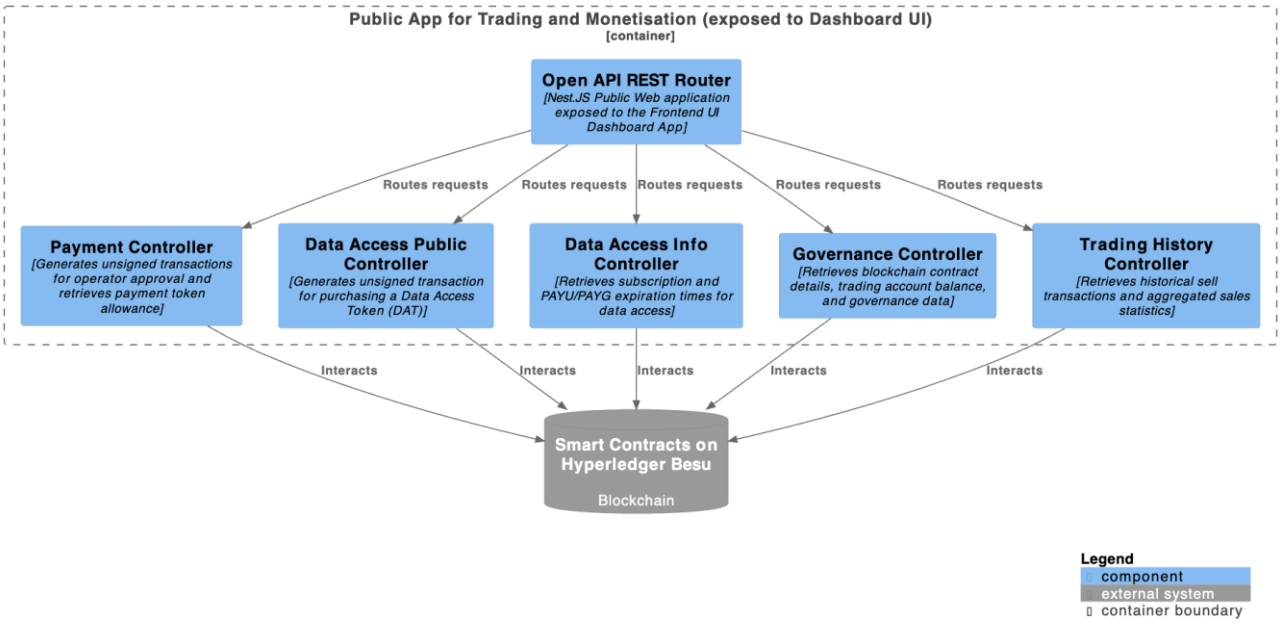

Figure 7 – Public App in Trading & Monetization Level 3 Component Diagram

The container, titled **Public App for T&M (exposed to Dashboard UI)**, is designed as a façade that exposes multiple public API endpoints. It serves as the intermediary layer between the front-end Dashboard User Interface and the blockchain system (specifically, Hyperledger Besu). The application leverages a modular architecture by decomposing its responsibilities across several controllers, each of which encapsulates a distinct set of functionalities related to payment processing, data access, governance, and trading history management. This modular design enables scalability, maintainability, and the clear separation of concerns. The container orchestrates the overall flow of requests through a central routing component, ensuring that incoming API calls are dispatched to the appropriate controller, where business logic is executed and blockchain interactions are initiated.

The container consists of following main components:

- **Open API REST Router** (Next JS/ Express JS Component): This component acts as the primary entry point for external API requests from the Dashboard UI. Implemented using the Nest.JS framework, the router encapsulates the routing logic necessary for dispatching client requests to various controllers. Its role is pivotal in ensuring that requests are correctly mapped based on operation identifiers and endpoint definitions outlined in the API specification. In doing so, it promotes a decoupled system design where the front-end does not require direct knowledge of the underlying controller implementations.
- **Payment Controller:** The Payment Controller is responsible for managing all payment-related operations. Its two main functions include generating unsigned transactions for the approval of an operator to spend payment tokens and retrieving the approved spending limits (allowances) of these tokens. This controller interacts directly with the Hyperledger Besu smart contracts to perform these operations, thereby ensuring that the business logic governing payment transactions adheres to the security and transparency requirements of blockchain-based systems. By abstracting payment token operations into a single component, the design ensures that payment processing remains both centralized and robust.
- **Data Access Public Controller:** The Data Access Public Controller handles the submission of purchasing orders for data access tokens. Its primary function is to generate unsigned transactions that, once signed by the user, permit the purchase of a Data Access Token (DAT). This transaction is essential for the monetization aspect of the platform, as it governs the mechanism through which users secure access rights to proprietary data. The controller validates the purchase access right DTO (Data Transfer Object) and interacts with the blockchain to facilitate secure, immutable transaction records. This separation of purchasing logic from other data access functionalities promotes clarity and ensures that each controller's responsibilities are narrowly defined.
- **Data Access Info Controller:** Complementing the purchasing capabilities of the Data Access Public Controller, the Data Access Info Controller is tasked with retrieving crucial information about data access subscriptions. Specifically, it offers endpoints that provide subscription expiration times in various contexts, including regular subscriptions as well as PAYU (Payment Upfront) and PAYG (Pay-as-you-go) scenarios. By furnishing users with the expiration details of their data access rights, this controller plays a key role in the post-purchase experience, enabling users to manage their access periods effectively. The controller's interactions with the blockchain ensure that the expiration data is retrieved from an immutable ledger, thereby maintaining the integrity and trustworthiness of the information provided.
- **Governance Controller:** The Governance Controller serves as a gateway to the system's blockchain governance functionalities. It retrieves critical information such as the latest

deployed contract addresses, diamond metadata (including contract-specific details), and trading account balances. Additionally, it supports queries related to the trading account's payment token balance. This component is central to ensuring that users and administrators have real-time access to the operational parameters and governance state of the system. By providing endpoints for both contract metadata and balance information, the Governance Controller underpins transparency and auditability, which are essential principles in blockchain-based environments.

- **Trading History Controller:** The Trading History Controller is designed to manage endpoints that retrieve historical trading data. Its responsibilities encompass two major areas: the extraction of individual historical sell transactions and the aggregation of sales statistics. The aggregated data is presented based on various dimensions, such as asset identifiers, offering identifiers, and time intervals, enabling users to perform detailed analytical assessments of trading performance. The controller supports complex query parameters (such as filtering by offering IDs or asset IDs) and returns both granular transaction data and summarized statistics. This component is critical for market analysis, risk assessment, and strategic decision-making by providing a comprehensive view of past trading activities.

To further enhance security and system stability, the Public API enforces robust measures throughout its architecture. **JWT authentication** is required for all protected endpoints, ensuring that only authenticated and authorized users can access sensitive operations. Additionally, **rate limiting** is applied to control the number of requests per client, effectively mitigating the risks associated with excessive traffic and potential abuse. Crucially, the system employs a user-controlled transaction signing mechanism, meaning that no blockchain operation is executed without explicit user authorization. These integrated security protocols not only safeguard the system against common vulnerabilities but also contribute to the overall reliability and trustworthiness of the platform.

All controllers within the container maintain a direct relationship with the external blockchain system, specifically Hyperledger Besu. This blockchain integration ensures that all transactions – be it for payment processing, data access orders, governance inquiries, or trading history retrieval – are executed in an environment that upholds immutability, transparency, and security. By standardizing interactions through this common external system, the architecture not only simplifies the development process but also enhances the overall trustworthiness of the application.

2.1.3.3. Smart Contracts in T&M System

The Trading & Monetization System relies on a robust set of smart contracts that facilitate tokenized transactions, governance mechanisms, and marketplace interactions. These contracts are deployed on the Hyperledger Besu blockchain, ensuring security, transparency, and automation in the exchange of data assets.

To maintain flexibility and long-term scalability, the system incorporates **upgradeability mechanisms**, allowing contracts to evolve without disrupting existing functionality. The upgradeability patterns used in the Trading & Monetization System ensure that contract logic can be modified while preserving state and user interactions.

2.1.3.4. Upgradeability Patterns in the Trading & Monetization System

Smart contracts in the system follow two different **upgradeability patterns**, depending on their role and complexity:

- **Transparent Proxy Pattern:** The Payment Token Contract (ERC-20) and Data Access Token Contracts (ERC-1155) follow the Transparent Proxy Pattern, a widely used upgradeability mechanism that separates contract logic from stored data. In this model, a proxy contract maintains all contract storage and delegates execution to an implementation **contract**, which can be upgraded as needed. This allows token-related functionalities such as minting, burning, and transferring tokens to evolve over time without affecting the token balances of users.
- **Diamond Standard (EIP-2535):** More complex contracts, such as Trading Management, Bourse, and Governance, leverage the Diamond Standard (EIP-2535) to enable modular and scalable contract architectures. Instead of relying on a single large contract, these contracts use facets, which are individual contract modules that handle specific logic. The Diamond Proxy Contract (Diamond.sol) manages these facets, allowing new functionalities to be added, modified, or removed dynamically without disrupting the core system.

Beyond trading, other critical components of the system also follow the Diamond Standard, including the Bourse Contract and the Governance Contract.

2.1.3.5. Trading Manager Contract

The Trading Manager Contract is a core component of the Trading & Monetization System. It orchestrates the marketplace's business logic, enabling interactions between buyers, sellers, and smart contracts, and is responsible for handling data access purchases, token transactions, and marketplace interactions.

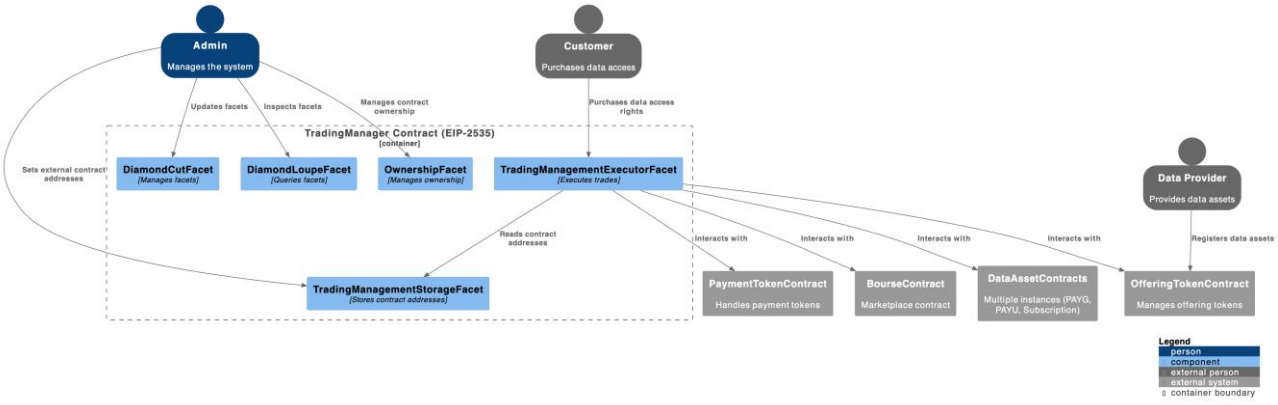

Figure 8 – Trading Manager Contract in Trading & Monetization Level 3 Component Diagram

Unlike traditional monolithic smart contracts, which can be difficult to upgrade and expand, the Diamond Standard (EIP-2535) allows the Trading Manager Contract to be split into separate facets. Each facet encapsulates a specific part of the contract's logic, making it easier to update, extend, or remove functionalities without disrupting the entire system.

The Trading Manager Contract consists of the following **facets**:

- **DiamondCutFacet**: This component manages contract upgrades by enabling the addition, removal, or replacement of facets. It allows the system administrator to modify the business logic over time while keeping the contract state intact.
- **DiamondLoupeFacet**: This facet provides introspection capabilities, allowing anyone to query which facets are attached to the contract. It helps external systems and developers understand the current structure of the Trading Manager Contract.

- **OwnershipFacet**: This facet manages contract ownership and administrative controls. The system administrator can transfer ownership, renounce ownership, or assign new administrators as needed.
- **TradingManagementStorageFacet**: This facet acts as a storage registry, maintaining references to external contract addresses, including the Bourse Contract, Payment Token Contract, and Data Asset Contracts. By keeping external contract addresses in a separate facet, the system ensures modularity and reconfigurability.
- **TradingManagementExecutorFacet**: This is the core functional component of the contract, responsible for executing trades and managing transactions. It processes data access purchases, interacts with offering and payment contracts, and ensures that the correct tokens and assets are exchanged.

The above description of Trading Manager Contract serves as a real-world example of how EIP-2535 can be leveraged to create a flexible, evolving smart contract system. Future contract descriptions will focus on business logic and interactions, but this section establishes the fundamental architectural approach that underpins also other contracts in the Trading & Monetization System.

2.1.3.6. Bourse Contract

The **Bourse Contract** constitutes the trading layer responsible for executing and managing the exchange of tokens within the system. It is invoked by an external Trading Manager Contract that oversees overall trading business logic.

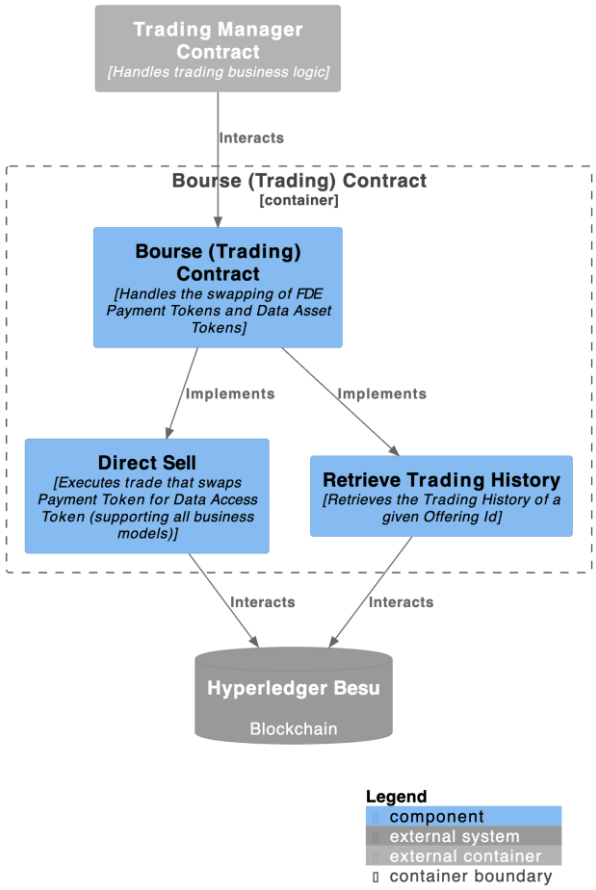

Figure 9 – Bourse Contract in Trading & Monetization Level 3 Component Diagram

- **Bourse Component:** This component orchestrates the swapping operations between FDE Payment Tokens and Data Asset Tokens. It is designed to support the internal mechanics of token exchange, ensuring that trade execution aligns with the prescribed business rules.
- **Direct Sell:** Dedicated to executing trades, this sub-component handles direct exchanges where a Payment Token is swapped for a Data Access Token. It accommodates various business models by processing token swap transactions and interfacing with the blockchain to record the trade events.
- **Retrieve Trading History:** This sub-component provides read access to historical trading data based on a given Offering Id. It enables the retrieval of past trade records, thereby supporting auditability and analysis of trading activities stored in the blockchain.

2.1.3.7. Governance Contract

The Below diagram provides a detailed view of the **Governance Contract** container, breaking it down into its individual components.

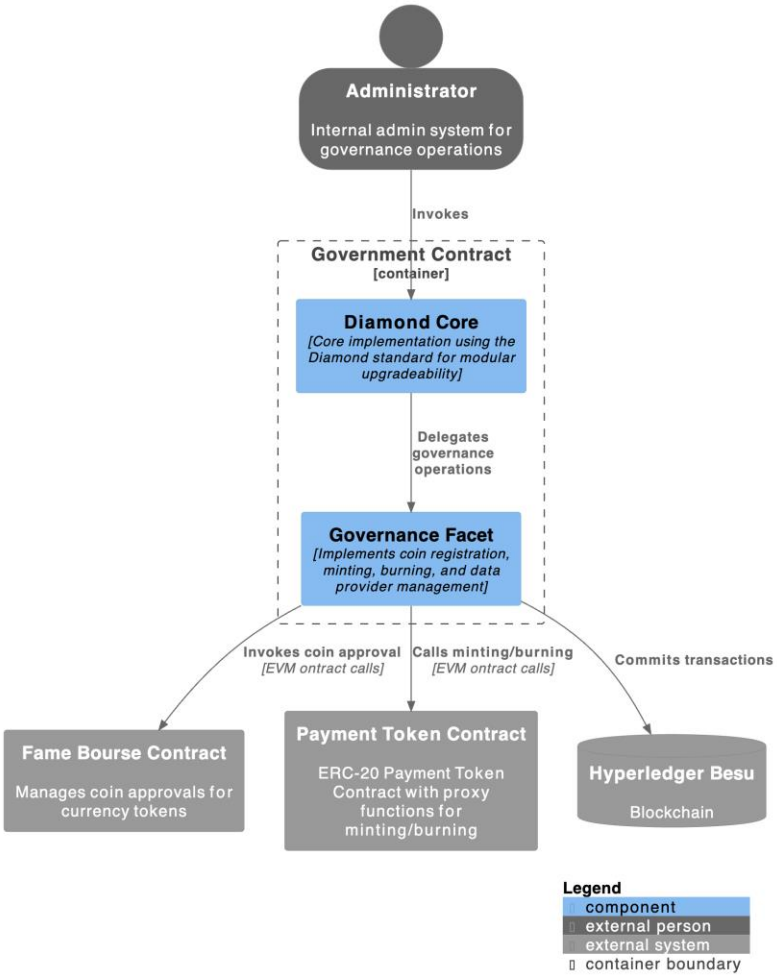

Figure 10 – Governance Contract in Trading & Monetization Level 3 Component Diagram

The Governance Contract is a central component responsible for managing coin registrations, token minting and burning, and data provider oversight within the ecosystem. It is architected using the

Diamond standard to ensure modular upgradeability and flexibility in incorporating new governance functionalities. The contract interacts with an internal administrative system to initiate operations, collaborates with external contracts for coin approval and token operations, and commits all transactions to the blockchain.

- **Diamond Core:** Implements the core functionalities using the Diamond standard, providing a modular framework that allows for incremental upgrades and the delegation of governance operations to specialized facets.
- **Governance Facet:** Realizes the primary governance functions including coin registration, minting, burning, and data provider management. It bridges interactions with the **FAME** Bourse Contract for coin approvals and with the Payment Token Contract for token transactions, while ensuring that all operations are committed to the blockchain.

2.1.3.8. Escrow Contract

The diagram below provides a detailed view of the **Escrow Contract** container, breaking it down into its individual components.

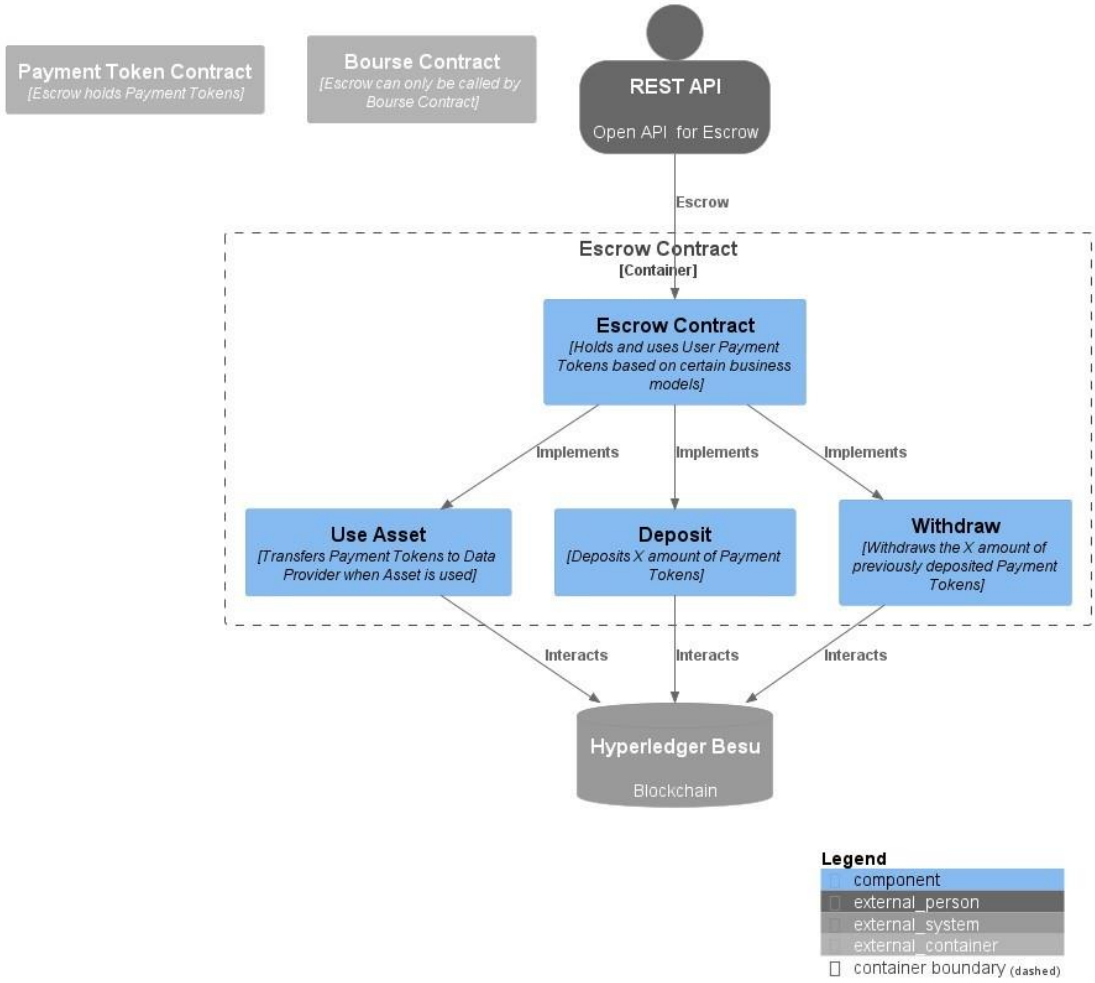

Figure 11 – Escrow Contract in Trading & Monetization Level 3 Component Diagram

The diagram includes the following components:

- **Escrow Contract**: This is the main component that handles the holding and use of the ERC-20 Payment Tokens.

- **Use Asset**: This component uses X amount of Payment Tokens based on the number of services used by the user and transfers those tokens to the beneficiary.
- **Deposit**: This component is used to deposit X amount of Payment Tokens to Escrow when purchasing access tokens.
- **Withdraw**: This component is used to withdraw X amount of Payment Tokens by the user.

The REST API interacts with the Escrow Contract component, which in turn implements the functionality of the Use Asset, Deposit and Withdraw components. The Use Asset and Deposit are called automatically based on user behavior and each of these components interacts with the Hyperledger Besu blockchain, which is an external system represented as a database in the diagram. The interactions with the blockchain involve various operations related to the specific functionality of each component.

2.1.3.9. Offering Contract

Error! Reference source not found. provides a detailed view of the **ERC-721 Offering Smart Contract** container, breaking it down into its individual components.

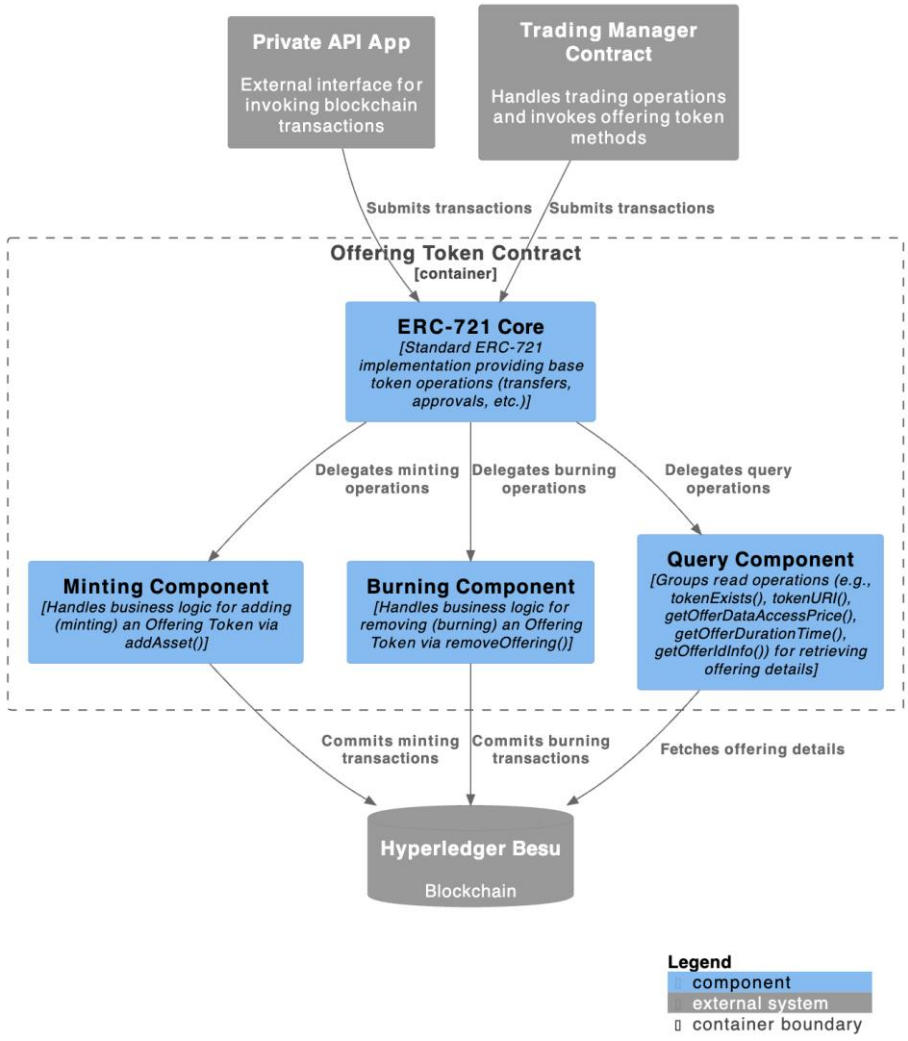

Figure 12 – Offering Token Contract in Trading & Monetization Level 3 Component Diagram

The Offering Token Contract is a smart contract module that manages offering tokens using the

ERC-721 standard. The contract is encapsulated within a bounded context that comprises several specialized components:

- **ERC721 Core:** This component implements the standard ERC721 functionalities, such as token transfers and approvals.
- **Minting Component:** This element is responsible for executing the business logic related to token creation through the *addAsset()* function. It validates input data and commits the resulting minting transactions to the blockchain.
- **Burning Component:**This component handles the logic for token removal via the removeOffering() function. It processes requests to burn tokens and ensures that these transactions are properly committed to the blockchain.
- **Query Component:** Grouping all read operations, this module provides methods such as *tokenExists()*, *tokenURI()*, *getOfferDataAccessPrice()*, *getOfferDurationTime()*, and *getOfferIdInfo()*. It fetches offering details from the blockchain to support data retrieval without altering the contract state.

2.1.3.10. Data Access Contracts

The **Data Access Token Smart Contracts** container is a core component of the tokenization platform, designed to support secure and scalable data access monetization. This container includes three distinct contract instances – **PAYG, PAYU, and Subscription** – each tailored for a specific business model to address diverse payment and usage scenarios. By leveraging the Transparent Upgrade pattern, these contracts can be updated seamlessly, ensuring that improvements or necessary patches can be applied without disrupting service or compromising security.

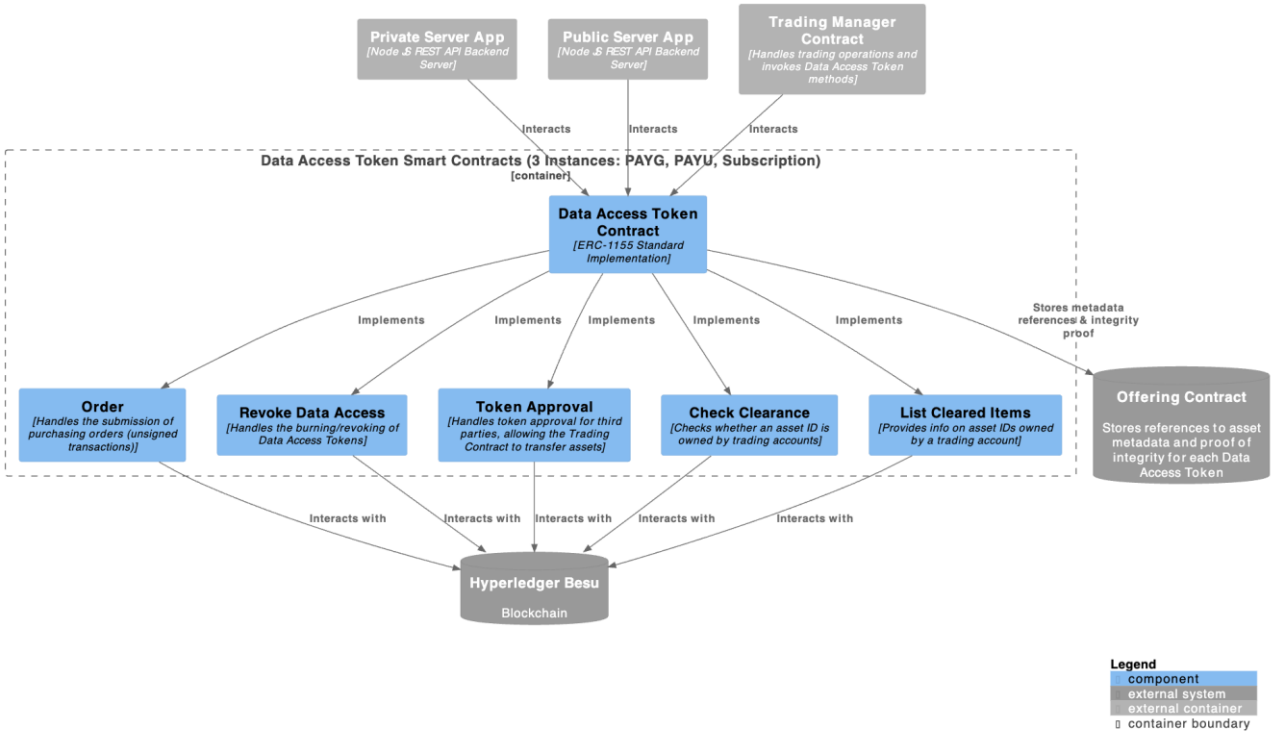

Figure 13 – Data Access Token Contract in T&M Level 3 Component Diagram

The above Figure 13 provides a detailed view of the ERC-1155 Data Access Token Smart Contract container, breaking it down into its individual components:

- **Data Access Token Contract (ERC1155 Standard Implementation):** Implements the ERC1155 standard to manage data access tokens. It serves as the core contract, facilitating subsequent operations and storing metadata references along with integrity proofs.
- **Order:** Handles the submission of purchasing orders through unsigned transactions. This component captures purchase intents for acquiring data access tokens.
- **Revoke Data Access:** Manages the burning or revocation of data access tokens. It processes token invalidation requests to remove token access rights on the blockchain.
- **Token Approval:** Facilitates third-party approvals, enabling the Trading Contract to transfer assets on behalf of token holders. This ensures that token transfers occur only with proper authorization.
- **Check Clearance:** Verifies whether a given asset ID is owned by designated trading accounts. It is critical for confirming token ownership before executing transactions.
- **List Cleared Items:** Provides information on asset IDs held by a specific trading account. This component supports the retrieval of token ownership details for audit and verification purposes.

2.1.3.11. Payment Contract

The Payment Token Contract is a blockchain-based smart contract built on the ERC20 standard. It is designed to manage payment tokens within the ecosystem by supporting functionalities such as token creation, destruction, transfers, balance inquiries, approval management, and supply tracking. This contract interacts with multiple external entities including a Private Server App, Trading Manager Contract, Escrow Contract, and Bourse Contract, all of which trigger token operations that are recorded on the blockchain.

The diagram below represents key contract elements:

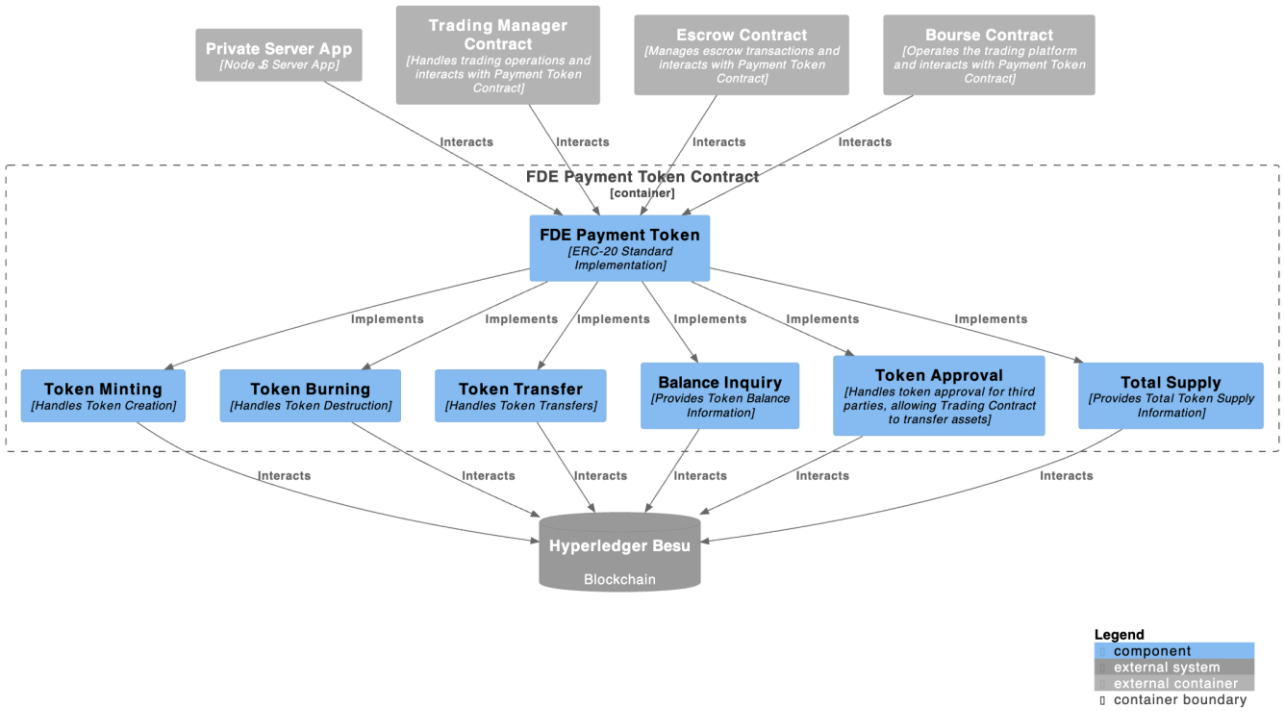

Figure 14 – Payment Token Contract in Trading & Monetization Level 3 Component Diagram

- **FDE Payment Token:** Implements the ERC20 standard to serve as the foundational token layer. This component encapsulates the core token operations, providing a standardized interface for managing FDE Payment Tokens.
- **Token Minting:** Handles the creation of new tokens. It executes minting operations and commits the resulting transactions to the blockchain, ensuring that new tokens are correctly recorded in the ledger.
- **Token Burning:** Manages the destruction of tokens from circulation. This component processes burning operations, securely removing tokens from the total supply on the blockchain.
- **Token Transfer:** Executes the transfer of tokens between accounts. It ensures that token movement is validated and recorded on the blockchain, adhering to the standard transfer protocols.
- **Balance Inquiry:** Provides real-time token balance information for specific addresses. This component interacts with the blockchain to retrieve accurate balance data, reflecting the current state of token holdings.
- **Token Approval:** Facilitates third-party token approvals. It allows designated contracts, such as the Bourse Contract or the Governance Contract, to transfer tokens on behalf of token holders, ensuring secure and authorized asset transfers.
- **Total Supply:** Computes and provides the total number of tokens in circulation. This component interacts with the blockchain to present up-to-date information on the overall token supply.

2.1.4. Code (Level 4) Architecture of T&M

This section focuses on the critical code-level architectural decisions that underpin the T&M system. Here, we present an overview of the most important design choices. Detailed code

documentation and implementation specifics are available in the repository's main *README.md* file, which should be consulted for an in-depth understanding of the codebase.

Monorepo Architecture for the Trading & Monetization System

The Trading & Monetization (T&M) system is structured as a **monorepo**, consolidating all code related to the private and public applications, as well as the smart contracts, within a single repository. This approach enables code reuse, unified dependency management, and a streamlined development workflow, ensuring consistency across all system components.

The monorepo is organized into multiple applications and shared libraries, each serving a distinct role within the system:

- **Private API Application**: Facilitates internal backend-to-backend interactions with smart contracts, allowing governance-related actions, offer management, and data access token administration. This application is not exposed to external users and is strictly secured.
- **Public API Application**: Provides a publicly accessible API that enables users and third-party services to interact with the system. It supports functionalities such as browsing marketplace offerings, purchasing data access tokens, and submitting blockchain transactions in a user-controlled manner.
- **Smart Contracts Library**: Contains all solidity-based contracts responsible for managing key blockchain interactions, including data access tokenization, payment processing, trading, governance, and escrow functionalities.
- **Shared API Utilities**: A common library that includes authentication mechanisms, blockchain utilities, and request-handling logic, ensuring that both applications adhere to a consistent implementation standard.

The monorepo follows a **containerized deployment approach**, with separate Docker images built for private and public applications. This ensures that the applications can be independently deployed and scaled, while maintaining compatibility with a shared codebase.

The monorepo includes **various auxiliary scripts** that assist in system management and development. Some notable utilities include:

- **NX Dependency Graph Visualization**: The repository is structured using **NX workspace**, and an automated dependency graph can be generated to illustrate how different applications and libraries depend on each other.
- **Blockchain Management Scripts**: Scripts to deploy, initialize, and interact with smart contracts in different environments (e.g., local, testnet, production).
- **API Testing and End-to-End Testing**: Automated scripts to validate system functionality using test suites for both APIs and smart contracts.

The following image illustrates the dependency structure of the monorepo, showcasing how applications and shared libraries are interconnected:

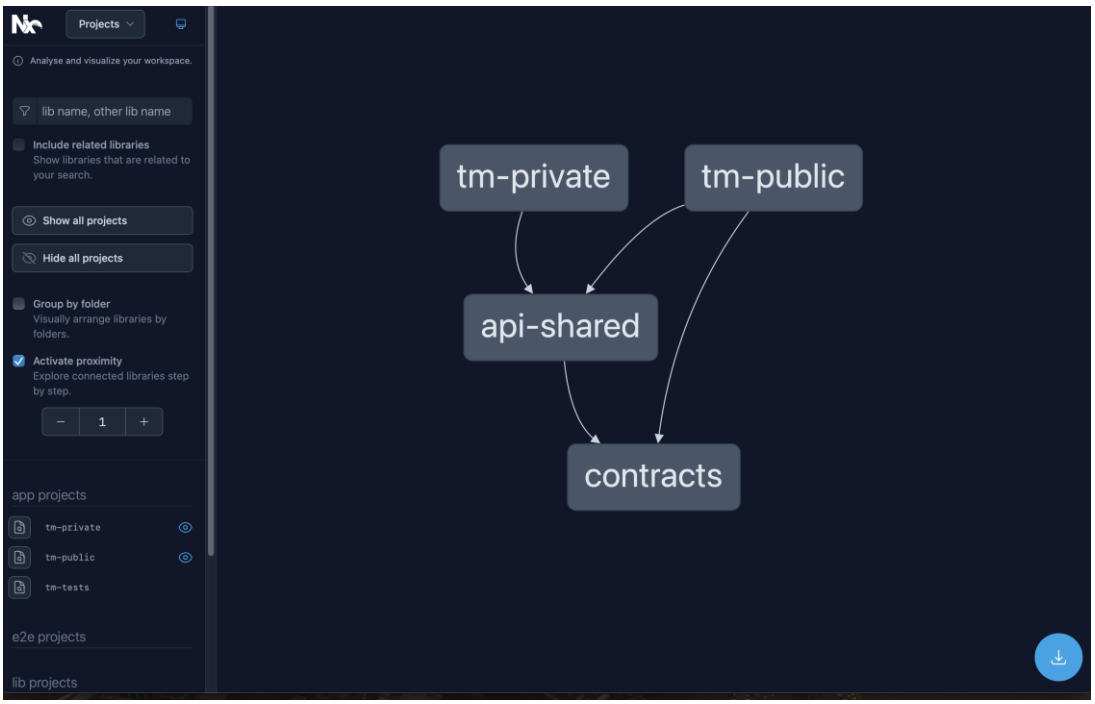

Figure 15 – Apps and Libraries Dependency Structure in Monorepo of Trading & Monetization

For detailed setup instructions, deployment guidelines, and development best practices, please refer to the **README.md** file within the monorepo.

2.1.5. APIs for Trade and Monetization System

This section offers a succinct overview of the API endpoints within the T&M module of the FAME project. The complete API documentation is automatically generated from the codebase definitions and is provided in Annex 1. We concentrate on the key endpoints that drive the most critical user journeys.

2.1.5.1. Setup trading

Description

Creates an Offering Token in the blockchain infrastructure that represents an offering.

Technical Specification

Receives a data structure (*Data Structures: Trading Setup Info*) that carries all the relevant information for initializing a blockchain trading environment for a given offering (see *P&T:Submit offering*). If the input is formally correct, it immediately returns confirmation to the caller. In the background, the module turns the input into a blockchain transaction that executes a smart contract.

The smart contract mints the Offering Token that represents the offering.

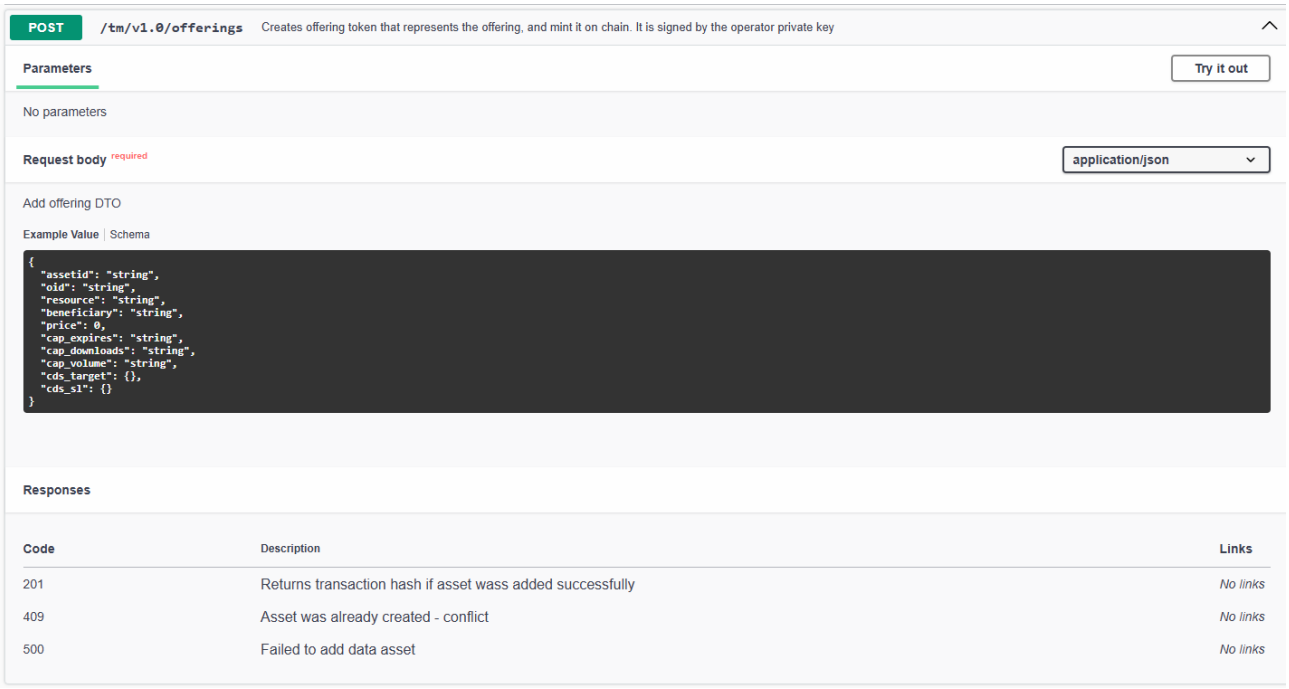

Figure 16 – Set up Trading API

2.1.5.2. Submit Purchasing Order

Description

Prepares and returns an unsigned transaction of a purchasing order for the interacting user, as illustrated in Figure 17.

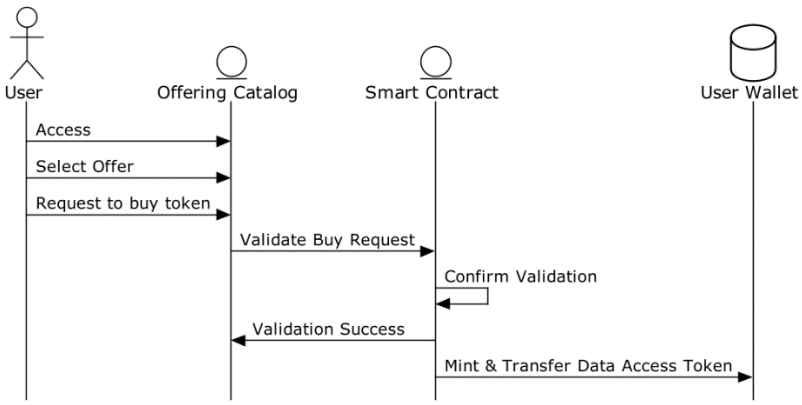

Figure 17 – Submit Purchasing Order Sequence Diagram

Technical Specification

This operation obtains, in a single user-signed blockchain transaction: A) the transfer of the correct amount of digital currency from an account owned by the user to an account owned by the publisher, and B) the transfer of a Data Access Token from an account owned by the FAME system to an account owned by the user.

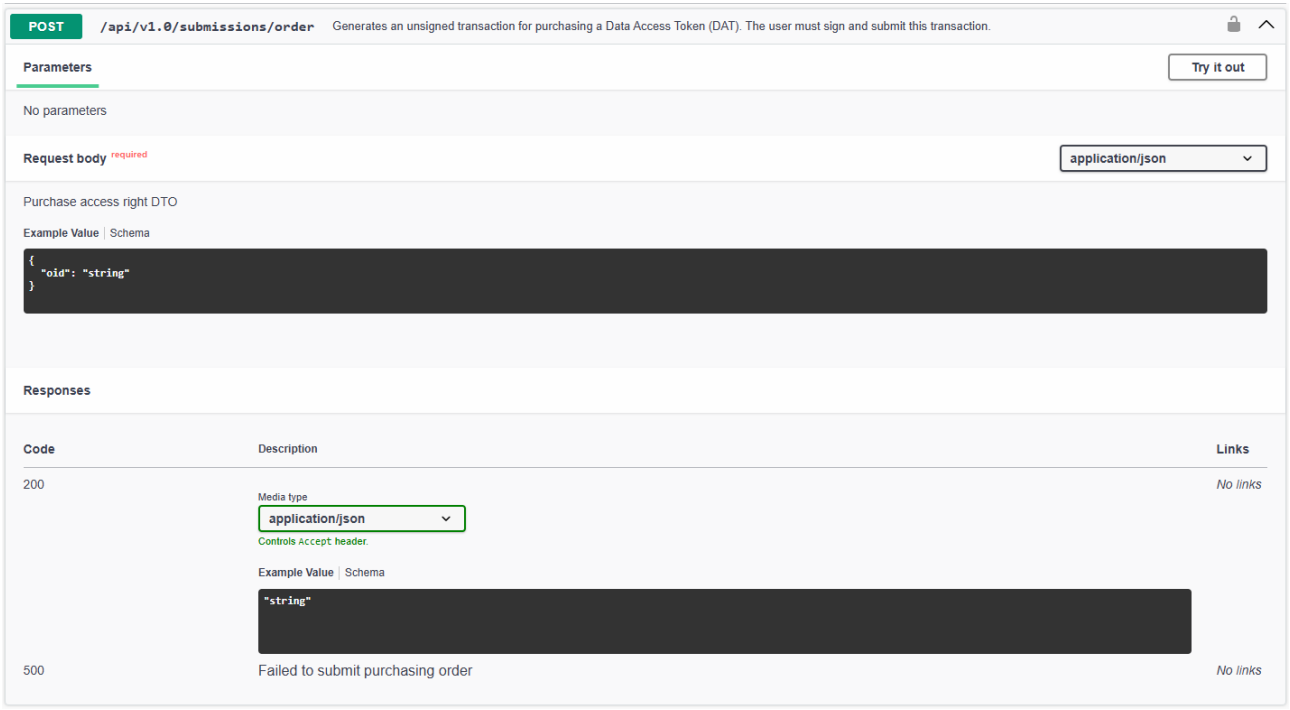

Figure 18 – Submit Purchasing Order API

2.1.5.3. Check Clearance

Description

Returns whether one or more trading accounts own a given asset identifier (AID).

Technical Specification

Receives an AID and a list of one or more trading accounts, each identified by its trade identifier (TID). If any of the given trading accounts is the owner of a currently valid access token linked to the given asset, it returns a confirmation.

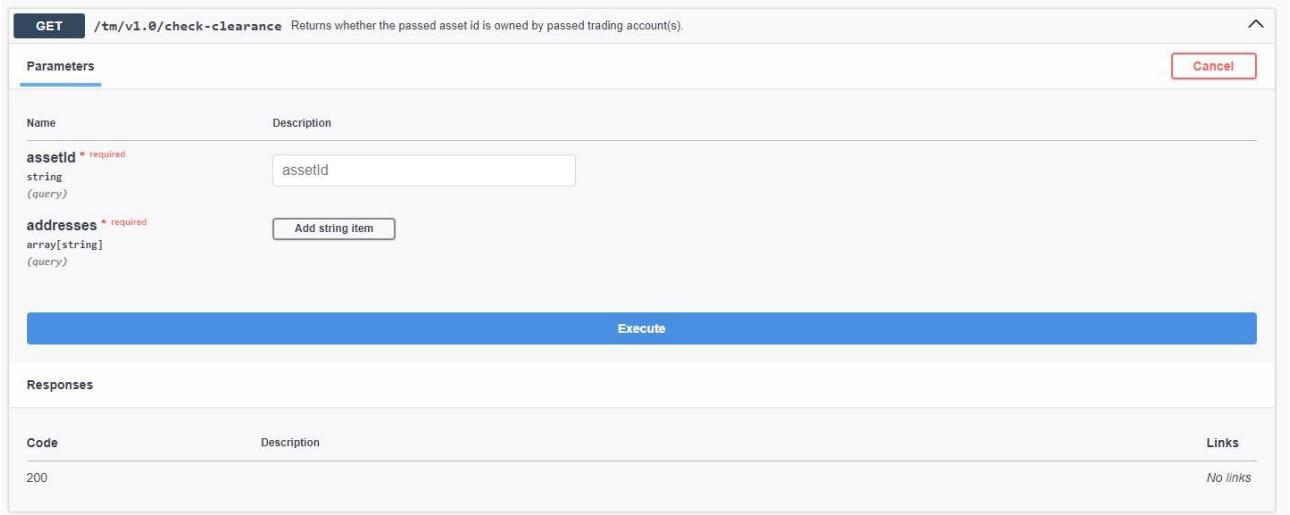

Figure 19 – Check Clearance API

2.1.5.4. List Cleared Items

Description

Returns all the owned AIDs of a given trading account.

Technical Specification

Receives the TID of a trading account. Returns the list of AIDs, if any, for which the given TID is the owner of a currently valid access token.

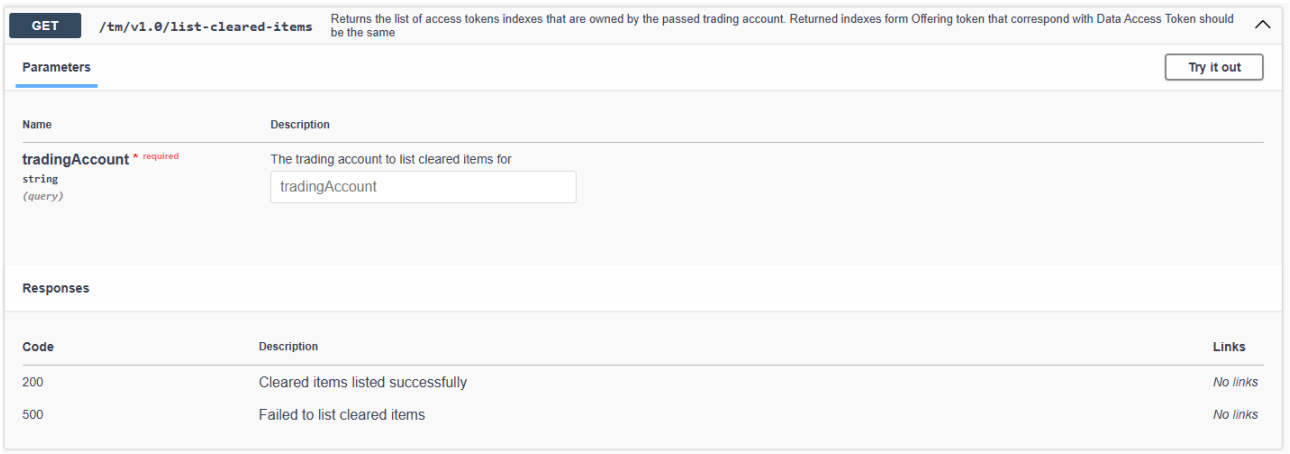

Figure 20 – List Cleared Items API

2.1.5.5. Retrieve Trading History

Description

The following API from the Bourse Contract is used to retrieve the trading history based on the passed information. Retrieve trading history related to selling and buying with the payment token.

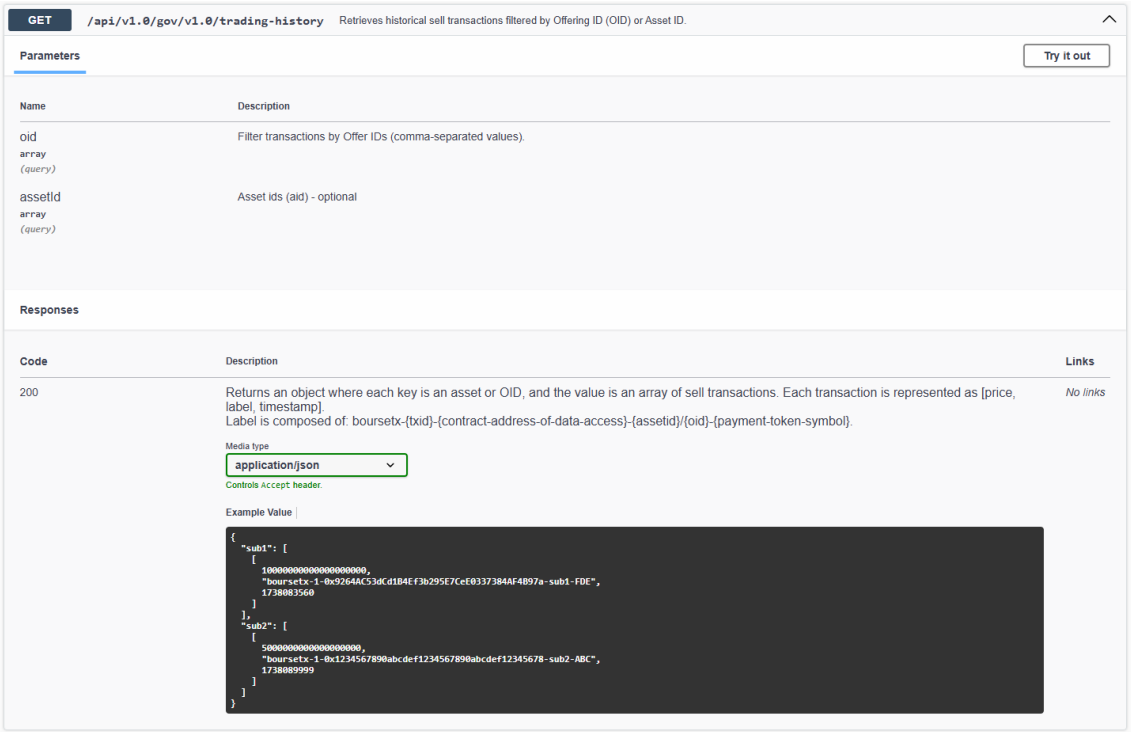

Figure 21 – Trading History API

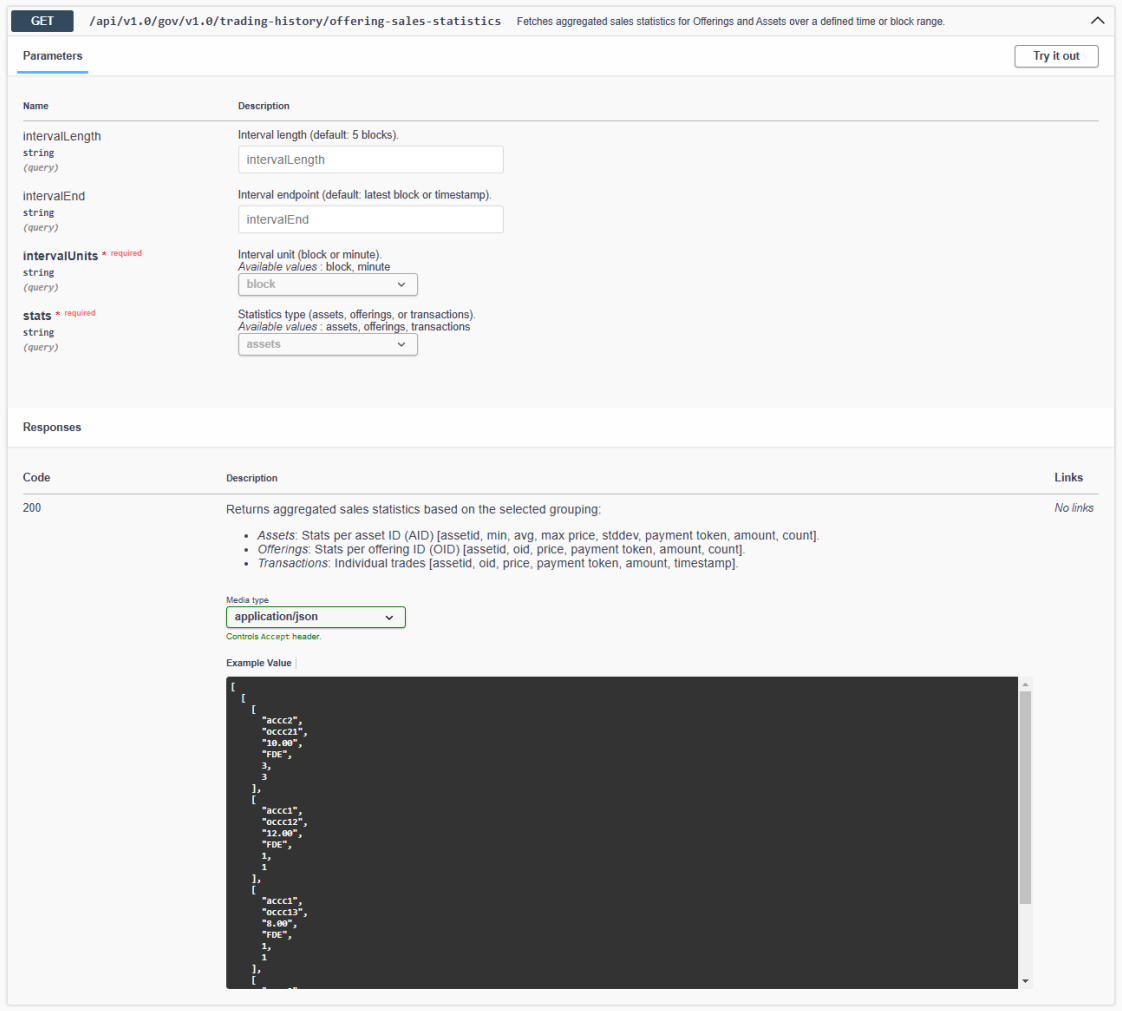

Figure 22 – Trading History Statistics API

2.2. Price Advising

FAME is implementing a data-driven pricing advisory mechanism leveraging issuer-provided data, stakeholder survey responses, and historical pricing realization analytics. Ultimately, the strategy intends to incorporate business interest-based metrics derived from the quantification of transactions of analogous assets. Nonetheless, the determination of asset pricing in the FAME ecosystem remains the prerogative of the client, as explicitly articulated within the Asset Offering documentation.

2.2.1. C4 Component-level Architecture

Figure 23 presents an overview of the price advising process, its users, and their interactions.

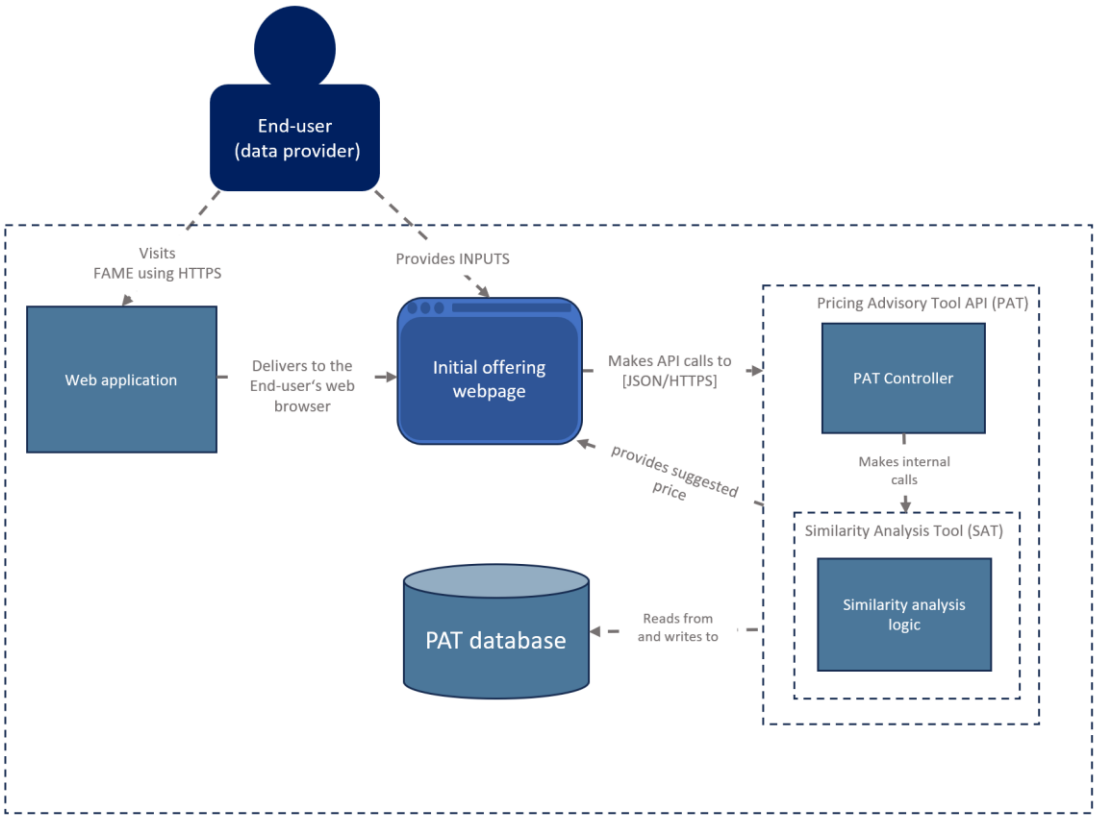

Figure 23 – Price Advising C4 Component-level Architecture

2.2.2. API for Pricing Advisory Tool (PAT)

Description

The Pricing Advisory Tool, known as the PAT, is a RESTful open API that offers price recommendations for assets and digital products based on user inputs. It aims to extract both subjective and objective influences impacting on the final price. Additionally, it provides the functionality to offer price ranges based on similar assets for which prices have historically been determined. Therefore, the end-user could view the potential price range within which the price of an asset or digital product may fluctuate (Figure 24).

Technical Specification

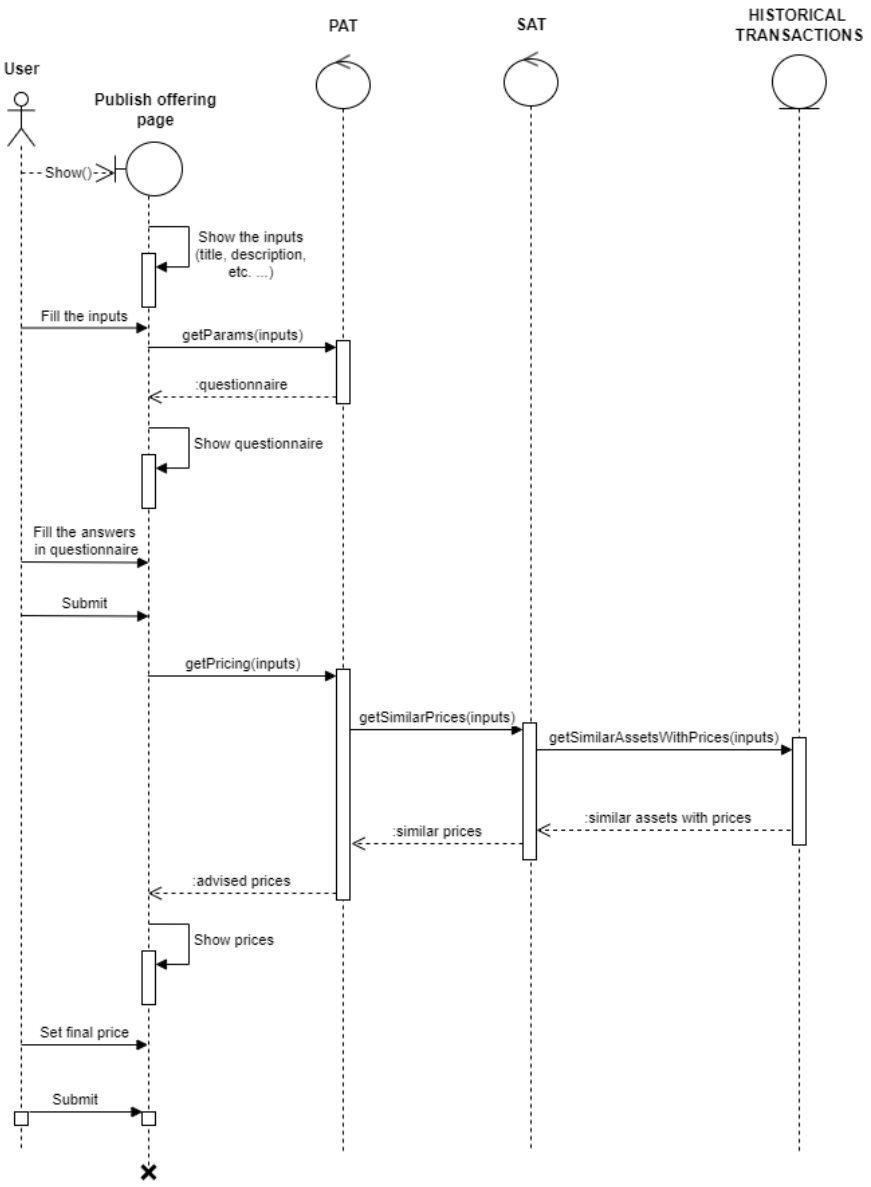

Figure 24 – Price Advisory Sequence Diagram

The publish offering screen directly utilizes the REST open API of PAT. Based on end-user inputs, a GET request is issued to retrieve questionnaire questions upon clicking the PAT button.

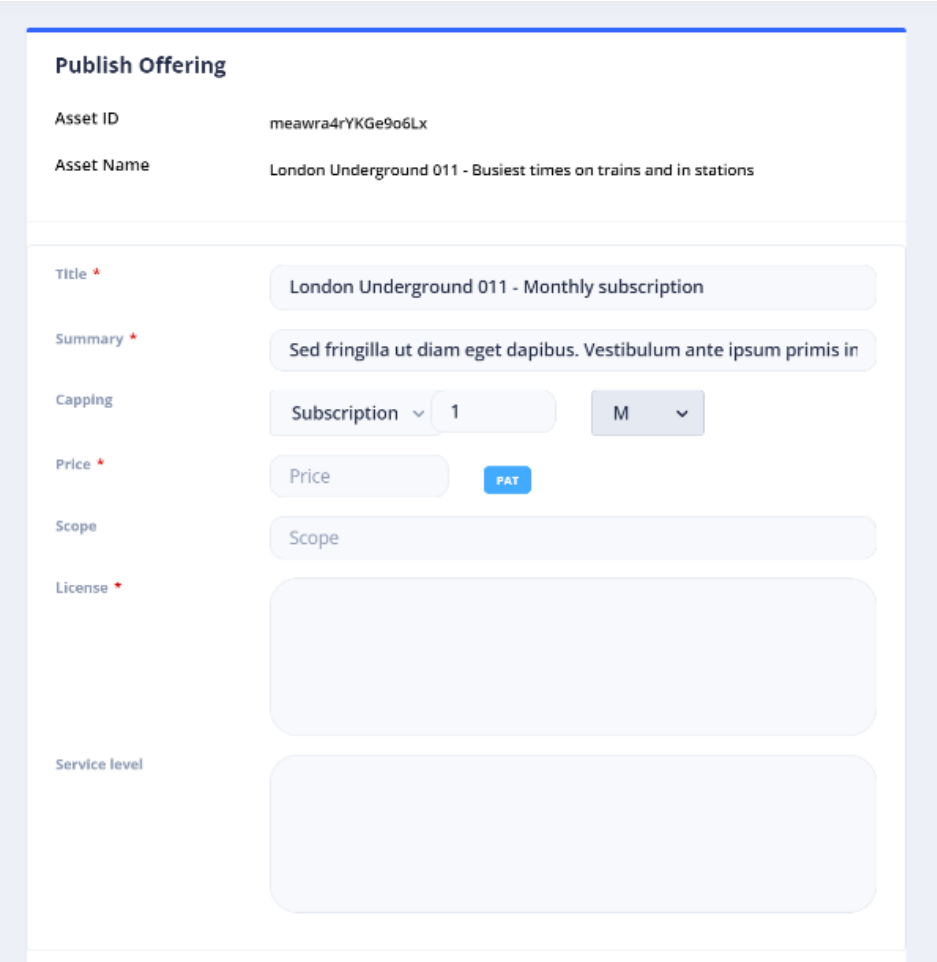

Figure 25 – Publish Offering Screen

The calculation flow for an asset's price proceeds through multiple phases:

1. At the outset, an asset-type-specific questionnaire is provided to the end-user, with the questions tailored based on the nature of the asset in question.
2. System retrieves responses from users through a REST API.
3. Based on the responses and data available about the asset, information is provided to the SAT interface responsible for identifying asset similarities – this process is contingent on the existence of preceding assets.

   Step 1 – get a subset of all datasets traded in the market

   Here, we will get a subset S of conceptually/contextually/domain-wise similar data assets. We will use the SBERT tool at this point – the similarity should be greater than 0.7. So, the set S will comprise data assets that are similar to the focal data assets in terms of their domain.

   Step 2 – create network of the set S

   To create network, we have to define edges. For this we need to compute matrix of similarities between all data assets in set S based on the questionnaire answers, i.e. QBS. The edge between two data assets in the set S will exist if their QBS (Questionnaire Based Similarity) is greater than 0.5. So, this is how the network N will be created.

   Step 3 – compute eigenvector centrality

   Eigenvector centrality of the network N will be computed.

Subsequently, a call is made to analyse the trading history for realization prices of similar assets - this step is contingent on the existence of preceding assets.

Step 4 – the price P2 will be computed

For this calculation, only the prices of connected assets (nodes/vertices) in network N will be used (these are data assets with QBS similarity higher than 0.5).

4. Upon acquiring all necessary inputs, the calculation of the recommended price for the end-user is conducted.

Asset-type-specific questions with various response types are sent back to the front-end, where they are displayed to the end-user.

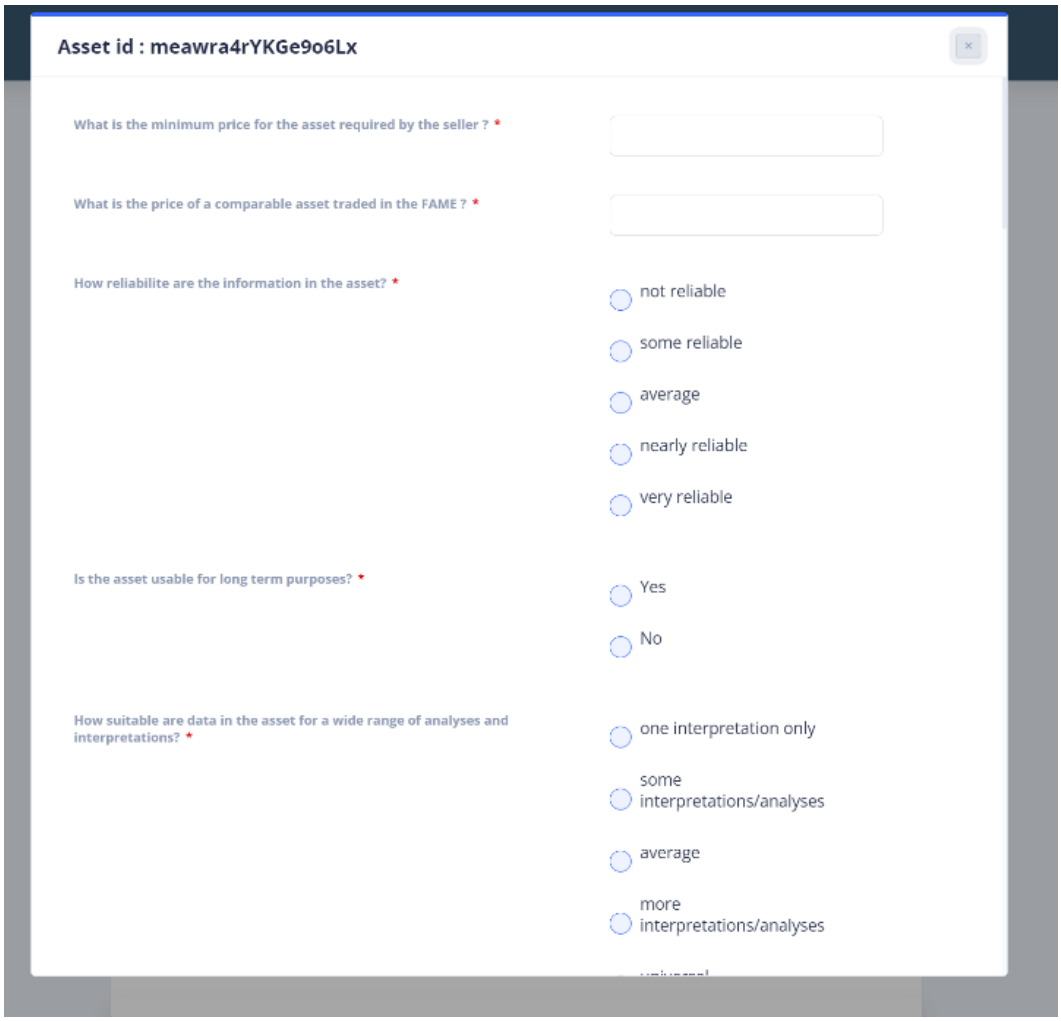

Figure 26 – Asset-type Specific Questionnaire

After being completed by the user, the responses are extracted and sent back to PAT for the computation of the recommended asset price. Upon executing the calculation logic, the recommended price is displayed to the end-user via the front-end. The user then has the option to utilize the suggested price or enter their own.

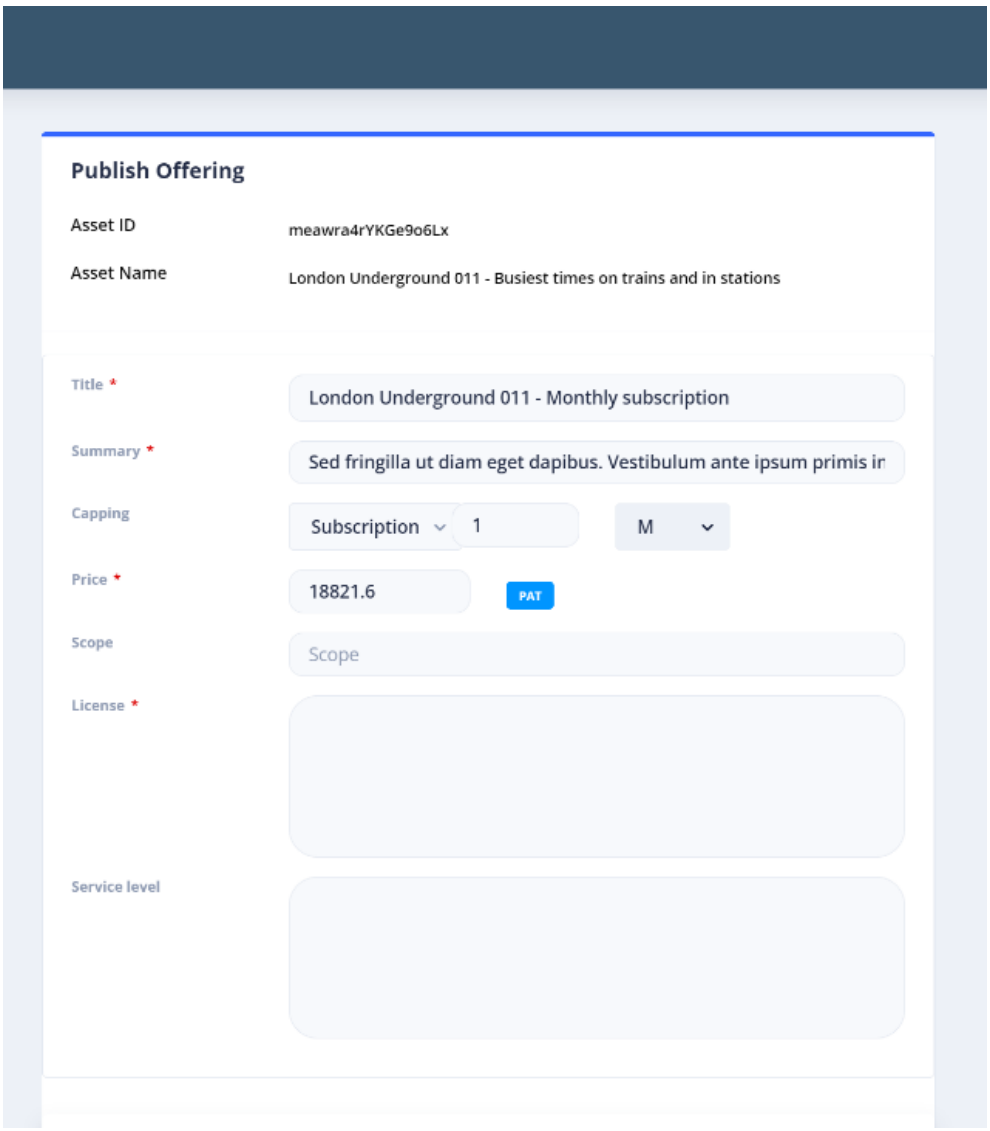

Figure 27 – Price Recommendation Interface

Table 1 – List of suggested questions for pricing data collection

0		What is the asset like?		Dataset		Digital Product
QG	Rank	Question Datasets	Questions digital product	Type of answer		
				Yes/No	Numeric	Points (categorical/ Likert scale)
1	1	Will the information in the asset be regularly updated?	Will the information in the digital product regularly updated?	x		
1	2	How suitable are data in the asset for a wide range of analyses and interpretations?				x

1	3	Is it possible to manage/read/update the data without additional software?	Is it possible to manage/read/update the digital product without additional software?	x		
1	4		What computational power is required to process the digital product?			x
2	1	Is the data clean? (1 if yes, 0 if it is needed to clean the data (redundancy, errors, typos...)?		x		
2	2	How old is the information in the dataset (months)?	How long ago has the digital product most recently been reviewed? (months)?		x	
2	3	When was dataset created/compiled (months)?	How old is the digital product (month)?		x	
2	4	Are the data in the dataset manipulated (adjusted) in any way? (Is the data raw?)		x		
2	5	Does dataset contain unique identifier so that it can be connected to other available datasets?		x		
2	6	How credible is the source of the asset?	How credible is the source/creator of the digital product?			x
2	7	Have these data been validated against independent sources or standards?	Has the digital product been validated against independent sources or standards?	x		
3	1	Is the dataset complete (i.e., without any missing information)? (Is maximum possible completeness achieved)?		x		
3	2	Is dataset clearly described (dictionary/metadata)?	Is the product supported by descriptive information (metadata, dictionary, subtitles, documentation)...	x		
4	1	How much resources were required to assemble and prepare the asset in question (in MD)?	How much resources were required to assemble and prepare the digital product in question (in MD)?		x	
4	2	What is the estimated number of customers?	What is the estimated number of customers?			
4	3	How many information points does the asset contain?			x	

4	4	How much capacity space is needed to store this data (in GB)?	How much capacity space is needed to store this digital product (in GB)?		x	
5	1	Is the dataset unique/original?	Is the digital product unique/original?			x
	2	Are there copyrights related to the data asset?	Are there copyrights related to the digital product?	x		
6	1	Was the renewable energy used in the process of asset creation?	Was the renewable energy used in the process of the product creation?	x		
7	1		What is the digital product type (8 categories)?			x

Table 1 above summarizes specific questions used to calculate intrinsic value of the digital asset (this is value-based pricing mechanism) for two broad groups: datasets and digital products. The value of the digital asset is based on the several dimensions – estimated cost for the customer, quality measured by: accuracy and validity of digital asset and completeness of digital asset; costs of creation, volume of customers, uniqueness/rarity, environmental sustainability (CO2).
For this phase, we use PAT as a tool to navigate business offering creator in setting price by calculating value of the asset **based on the cost-based approach and prices of similar assets. We use demand (represented by the number of transactions of similar assets) in price suggestion calculation**.

2.2.3. Similarity Analysis Tool (SAT)

Description

The Similarity Analysis Tool (SAT) is part of PAT API which purpose is to search for similar assets for which historical sales have been executed, i.e., there must already exist completed sales of the given asset type. Inputs to SAT include responses from a questionnaire as well as information from the asset offering such as Title and Asset Description, Business Model selected for the asset. Analysis of human-readable text from the title and asset description by an AI-based model creates clusters of similar asset types which can be further used to perform similarity analysis be used to find relevant similar assets.

Formulation of SAT:

$$Sim(A_i, A_j) = \frac{1}{3}\left(\frac{\sum_{k=1}^{k=l} w_k(A_i^k == A_j^k)}{\sum_{k=1}^{k=l} w_k} + \frac{\sum_{r=1}^{m} w_r \log(A_i^r) \times w_r \log(A_j^r)}{\sqrt{\sum_{r=1}^{r=m}(w_r \log(A_i^r))^2} \times \sqrt{\sum_{r=1}^{r=m}(w_r \log(A_j^r))^2}} + \frac{\sum_{t=1}^{n} w_t A_i^t \times w_t A_j^t}{\sqrt{\sum_{t=1}^{t=n}(w_t A_i^t)^2} \times \sqrt{\sum_{t=1}^{t=n}(w_t A_j^t)^2}}\right)$$

Figure 28 – SAT Formulation

where *l, m,* and *n* are numbers of logical, continuous and ordinal variables, *A* denotes-values of the responses received on the items in questionnaire (Table 1), and *w* denote weights associated with each question.
As for the weights, there is not much guidance in the literature. Equal weights shall be used at the beginning which may further be refined based on domain expertise.

Note: This part will be implemented into the PAT and is not currently a part of it.

Technical Specification

The process occurs in two steps:

1. *Preprocessing and Clustering for Subgroup Identification:*

The initial phase leverages Natural Language Processing (NLP) techniques to extract and preprocess textual data from user-provided asset titles and descriptions within the Asset Offering interface. This information undergoes a clustering analysis employing machine learning algorithms to systematically classify assets into finely segmented subgroups. This classification is based on the semantic and contextual similarities within the asset metadata, facilitating the delineation of discrete asset clusters for subsequent analysis.

2. *Similarity Analysis within Identified Subgroups:*

Following the clustering phase, the second step initiates a similarity analysis leveraging advanced algorithmic comparisons within each identified subgroup. This analysis employs vector space modelling and a composite similarity metric (based on the concepts such as cosine similarity, hamming distance, and Euclidean distance) to quantitatively assess the closeness of each asset to the target asset under consideration. The objective is to pinpoint assets that exhibit the highest degrees of similarity to the target, based on the multidimensional feature space generated from the asset's descriptive metadata.

This structured, two-step methodology enables granular and precise identification of similar assets within a vast dataset, facilitating targeted asset comparisons and enhancing the efficacy of asset-related decision-making processes.

2.3. Semantic Search Engine

2.3.1. C4 Component-level Architecture

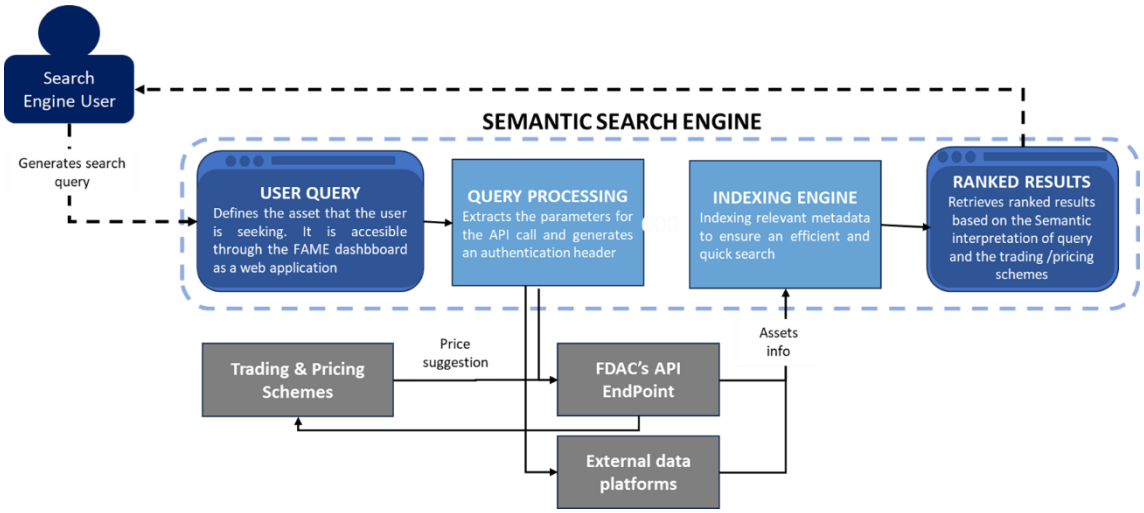

Figure 29 – Semantic Search Engine C4 Component-level Architecture

2.3.2. Components

1. **User Query**: This is where the search begins with the user input. The user defines what they're searching for using a textual query, an input sheet, or both. This input acts as the basis for generating a set of parameters that the search engine will use to retrieve relevant results from the FDAC.
2. **Query Processing**: In this stage, the search engine processes the user's input, extracting the search terms and any additional parameters necessary for the FDAC API call. It also generates any needed authentication header. The structured payload is then directed to the appropriate endpoint..
3. **Indexing Engine**: Here, the engine interacts with the FDAC's asset metadata. It indexes important information to enable efficient and quick searches. The indexing process is designed to accommodate expansions for new trading or pricing data as needed.
4. **Results**: The final output of the engine presents the user with ranked search results. It uses semantic analysis to interpret the user's query and combines this with the trading and pricing schemes to retrieve a list of assets that match the search criteria. The results are ordered

based on relevance to the query and can incorporate additional ranking logic when pricing or popularity data is available.

2.3.3. External Components

1. **FDAC's API Endpoint**: This component is the interface through which the search engine communicates with the FDAC. It sends structured requests to the FDAC to retrieve asset metadata that matches the user's search criteria. The endpoint now supports additional filters (e.g., asset type, developer, subscription model…) as well as an expanded or filter data mode..
2. **Trading & Pricing Schemes**: This component influences the final outcome by providing data such as price ranges or offering information that can affect the ranking of search results. Additionally, data gathered from the user interface—such as searches and asset interactions—may be leveraged by the PAT module to measure asset popularity. This information is recorded via two new endpoints for storing user queries and interactions, making it possible to refine or personalize search outcomes in the future.

2.3.3. API Endpoint Structure

The semantic search engine's API endpoint acts as an intermediary, formulating and directing queries to the FDAC at the updated path:

POST /api/data/search

When a user submits a query, the endpoint constructs a JSON request, with the following fields:

- **term**: the primary search string, directing the engine to retrieve assets related to it.
- **filters**: an array that refines search results further, specifying attributes like "owner" or "tag."
- **expand**: a boolean determining whether to return full JSON objects (true) or only IDs (false).
- **outputFilter**: designates which content types to return, such as "components" for data assets

.

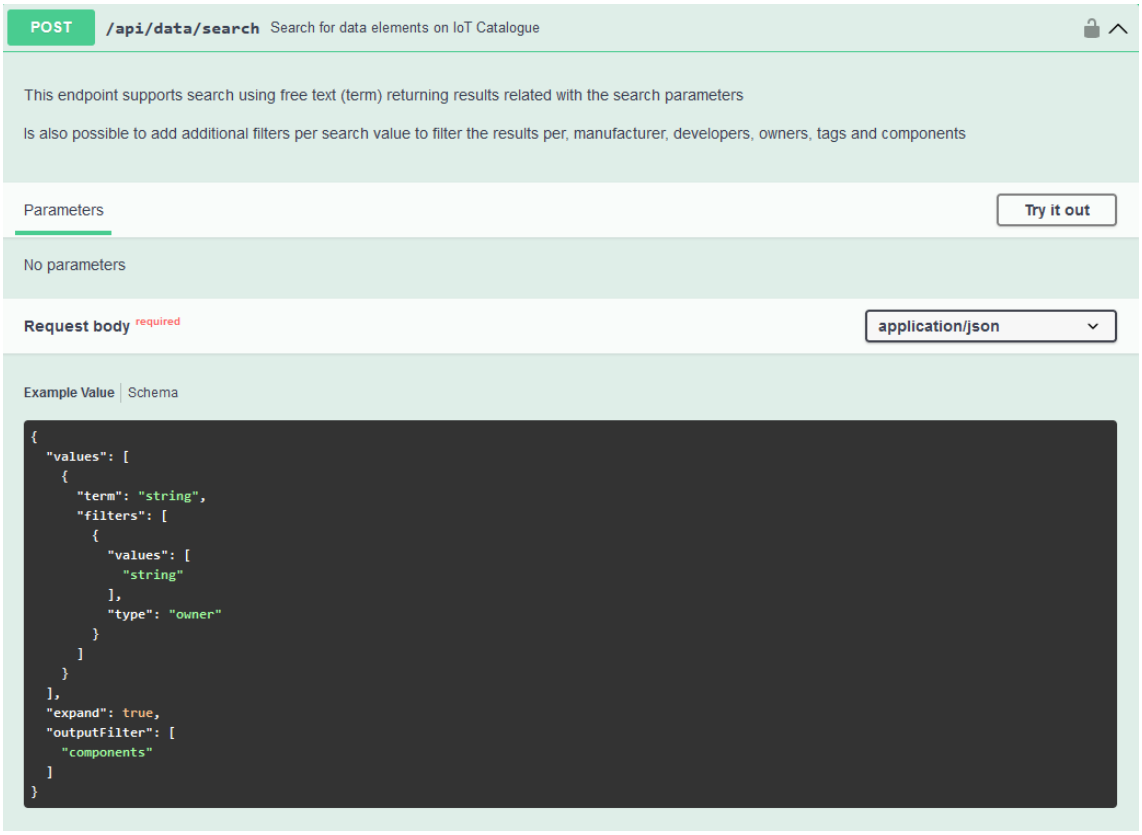

Figure 30 – Semantic Search Engine API endpoint

2.3.4. Integration with FDAC

The engine retrieves lists of data assets that align with the input search criteria. Assets are ranked based on semantic relevance, referencing metadata that includes names, summaries, tags, and owners. Developers' details, relevant imagery, and the visibility status are also considered to present a complete view of each asset. This approach supports both broad and specific queries within the FDAC's catalogue of data assets.

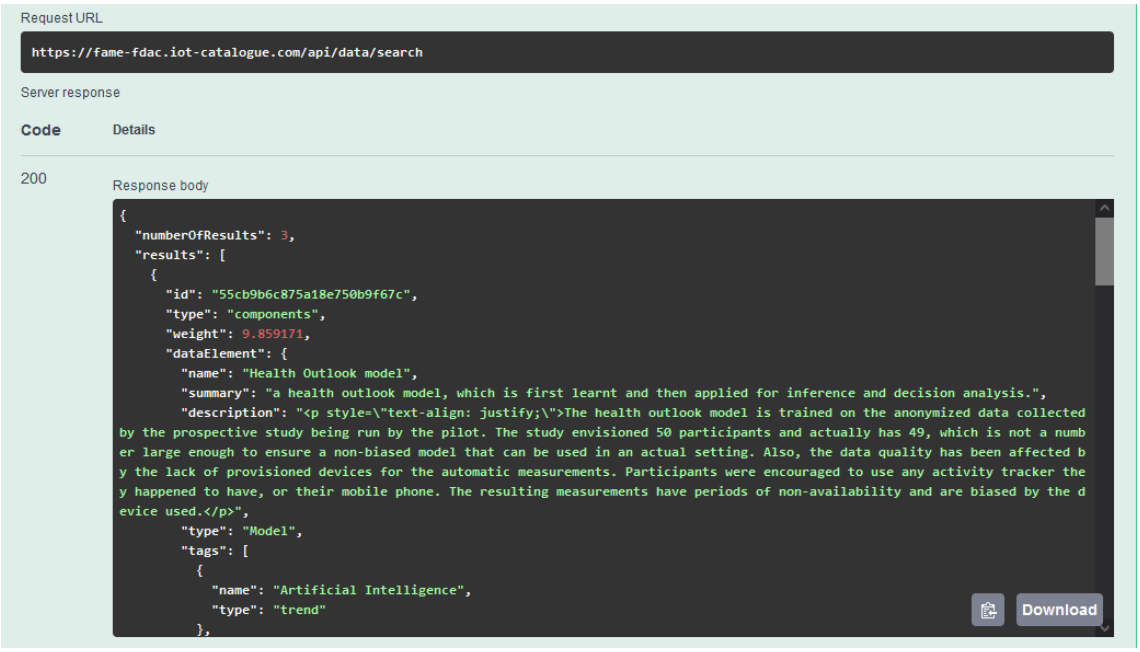

Figure 31 – Integrated SSE / FDAC API interface

2.3.5. State of PAT Integration

The integration with the PAT module has progressed to capturing user interactions for potential pricing and popularity analysis. Two endpoints are available to record data:

Search Queries

Column	Data Type	Description
search_id	SERIAL PRIMARY KEY	Unique identifier for the search.
query	TEXT	Text of the search entered by the user.
number_of_results	INTEGER	Total number of results returned.
filters	TEXT (JSON)	Filters applied to the search.
search_date	TIMESTAMP	Date and time of the search.

Asset Interactions

Column	Data Type	Description
interaction_id	SERIAL PRIMARY KEY	Unique identifier for the interaction.
search_id	INTEGER	Links to the search_queries table.
asset_id	VARCHAR(255)	Unique ID of the returned asset.

Column	Data Type	Description
asset_name	TEXT	Name of the asset (e.g., "Driving Profile AI Model").
weight	FLOAT	Weight of the asset in the results.
interaction_type	VARCHAR(50)	Can be "displayed" or "clicked."
interaction_date	TIMESTAMP	Date and time of the interaction.

	search_id	query	number_of_results	filters	search_date
1	1	data	5	{'price': [10, 10000]}	2025-03-04 14:56:31.000
2	2	AI	2	{'price': [10, 10000]}	2025-03-06 09:57:31.000
3	3	AI	2	{'price': [10, 10000]}	2025-03-06 11:19:55.000

	interaction_id	search_id	asset_id	asset_name	weight	interaction_type	interaction_date
1	1	1	5c3dc3770efe600de5613175	Qminer - Analytic platform for real-time large-sc...	3,18	displayed	2025-03-04 14:56:33.000
2	2	1	4aMZdcE8ELRneNpmw	Sample asset using interoperability API with DC...	4,24	displayed	2025-03-04 14:56:33.000
3	3	1	5c3dc3770efe600de5613164	Data Protection Orchestrator (DPO)	6,35	displayed	2025-03-04 14:56:33.000
4	4	1	848ed49deabaad8250b4f678	Blockchain addresses PageRank data	6,35	displayed	2025-03-04 14:56:33.000
5	5	1	6c4d66baa44015cd5e2ed63c	INFINITECH Data Collection	7,05	displayed	2025-03-04 14:56:33.000
6	7	2	e68a05c3ef4635b16a2925a0	Driving Profile AI Model	9,71	displayed	2025-03-06 09:57:32.000
7	8	2	5c3dc3770efe600de561315c	AI based detection of fraudulent immediate loan...	6,94	displayed	2025-03-06 09:57:32.000
8	9	2	5c3dc3770efe600de561315c	AI based detection of fraudulent immediate loan...	6,94	clicked	2025-03-06 09:57:55.000
9	10	3	5c3dc3770efe600de561315c	AI based detection of fraudulent immediate loan...	6,94	displayed	2025-03-06 11:19:56.000
10	11	3	e68a05c3ef4635b16a2925a0	Driving Profile AI Model	9,71	displayed	2025-03-06 11:19:56.000

Figure 32 – User Generated Data on Search Interface

By storing both the query details and interaction data, the search engine can later incorporate PAT's pricing information into result ranking, making it possible to highlight popular assets or those fitting certain price ranges.

2.3.6. Search Interface & User Journey

The semantic search interface remains with the same schema: users enter their search in the "Search Assets" box, optionally set filters (such as asset price range, expand mode, output filter, publisher, and type of asset), and trigger the search. The system then fetches and displays assets matching the user's criteria, with optional references to trading and pricing data where available.

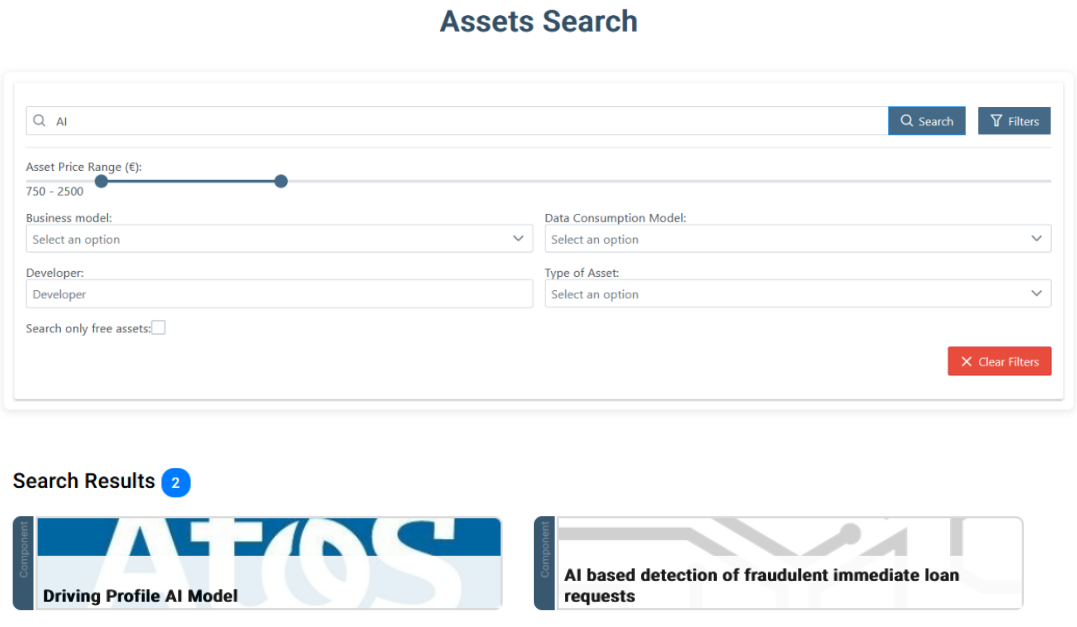

Figure 33 – Search Interface &Results

The FAME semantic search interface provides an intuitive user journey for discovering and selecting data assets. It begins with the input box titled "Search Assets" where users can enter keywords related to the assets they're looking for. Accompanying this are additional filtering options to refine their search:

1. **Asset Price Range**: A slider allows users to define a minimum and maximum price range, tailoring the search to fit their budget or value expectation.
2. **Expand**: A dropdown where users can choose to expand the details in the search results, typically toggled to 'True' for comprehensive information.
3. **Output Filter**: This dropdown enables users to refine the search output based on categories such as 'All', or specific types of assets within the FDAC.
4. **Publisher**: An input field where users can specify the owner or creator of the assets, helping them to find content from a preferred source.
5. **Type of Asset**: A dropdown menu that filters results according to the asset type, such as reports, datasets, models, etc., making the search more domain specific.

After setting the desired parameters, clicking the "Search" button triggers the system to fetch and display assets that match the user's criteria, aligning with both semantic relevance and pricing considerations. This design fosters a focused and efficient asset discovery process within the FAME environment.

2.3.7. Baseline Technologies in Use

Originally built with Flask for early rapid development, the search engine's backend has since evolved to incorporate Django for the data-collection endpoints. The frontend is managed by Angular and TypeScript for a dynamic and scalable user experience. SQL Server stores the search and interaction records. Communication with FDAC follows a RESTful API design using JSON, ensuring a structured and consistent data exchange. For deployment, Harbor, Kubernetes, and Argo CD handle image management, orchestration, and continuous delivery, respectively, aligning with project-wide DevOps practices.

2.4. Token Gateway System (TG)

As the FAME architecture evolved to support data publishing and tokenized access through the Trading & Monetization system, one important component was previously undefined: how users would *consume* the data they purchased.

To address this gap, the FAME consortium conducted a focused architectural study, exploring alternative models for integrating data access control and usage tracking. Two primary models were considered:

- **Option 1**: A *decentralized, Data Provider-managed infrastructure* where each provider retains full responsibility for access control and consumption validation.
- **Option 2**: A *shared infrastructure* managed by the FAME consortium, where FAME handles token validation, access control, and usage tracking centrally.

After careful evaluation of both approaches – based on system requirements, scalability, security implications, and provider autonomy – the consortium reached a consensus to adopt Option 1. The decision to move forward with such solution was driven by several strategic and technical factors:

- **Provider Control and Autonomy**: Option 1 ensures that Data Providers retain full control over their data assets, infrastructure, and access policies. This autonomy aligns with the

decentralized philosophy of the FAME ecosystem and respects the varying compliance, security, and operational requirements of different providers.

- **Security and Compliance**: By delegating sensitive access control and data delivery responsibilities to providers, this model reduces the security liability of the FAME consortium and allows providers to enforce bespoke, often domain-specific, security practices.
- **Minimal Overhead for FAME**: Option 1 keeps FAME's responsibility focused on blockchain-based access rights avoiding the need to manage large-scale data infrastructure, which would increase operational complexity and cost.
- **Modular and Scalable Architecture**: The approach naturally supports horizontal scaling, as each provider manages their infrastructure independently. It allows the system to grow organically with minimal coordination overhead and no single point of failure.
- **Alignment with Market Expectations**: Providers often prefer to maintain custody over their datasets, especially in sensitive domains like finance or healthcare. Option 1 aligns with these expectations by allowing them to remain in control, while still enabling standardized integration through FAME's token infrastructure.

2.4.1. System Context (Level 1) of TG

The high-level overview of the system, its users, and their interactions are depicted below:

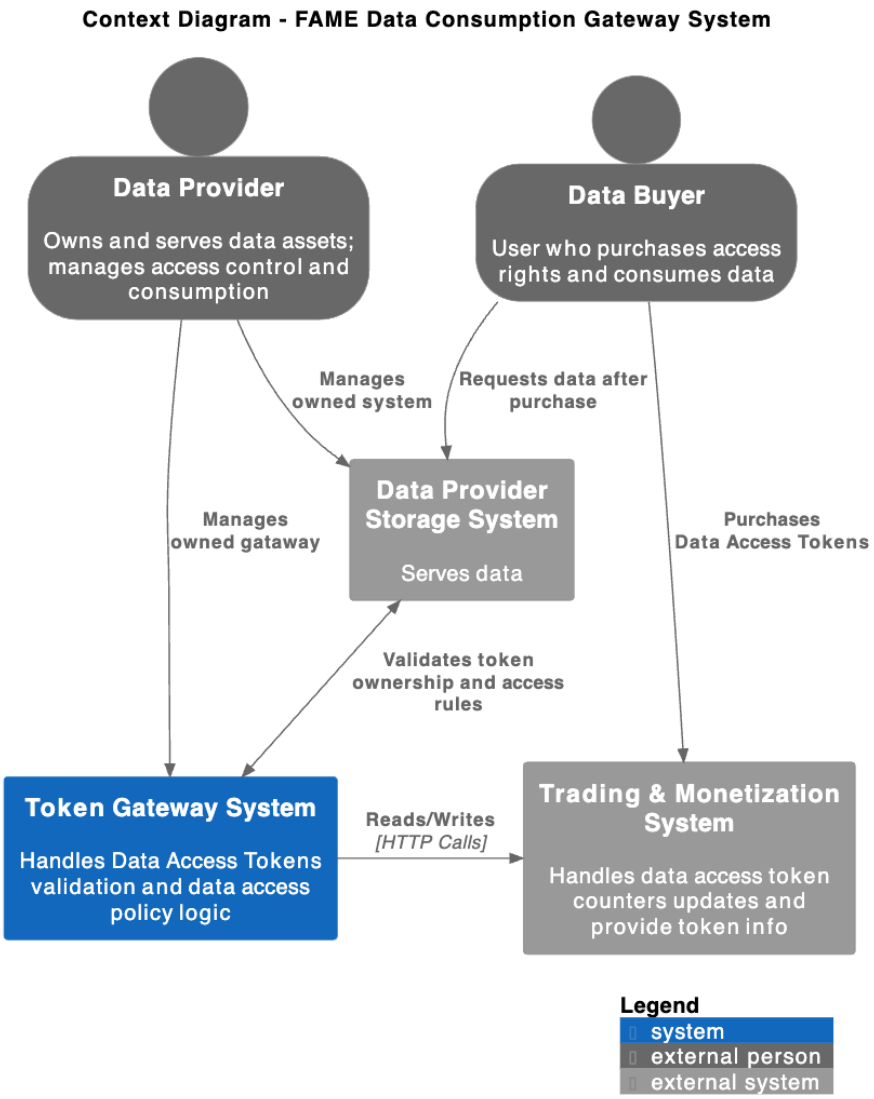

Figure 34 – Context Diagram for Token Gateway System

At the core of the data consumption process is the **Token Gateway System**, a component **implemented**, **deployed and managed by the Data Provider**. Its primary role is to validate Data Access Tokens (DATs) and enforce access policies that determine whether a Data Buyer is authorized to access a particular dataset.

After purchasing a DAT through the FAME Trading & Monetization System, the Data Buyer attempts to consume the data by sending a request to the Provider's Storage System. Rather than serving the data immediately, the storage system delegates the decision to the Token Gateway. This step ensures that access is only granted if the token is valid, and the usage conditions are met.

The Token Gateway validates the request by querying the Trading & Monetization System via HTTP or directly reads Data Access Token smart contract state. It fetches the relevant token metadata and current usage counters, which are used to determine if the access conditions – such as expiration dates, ownership, or usage quotas – are satisfied.

If the token is valid and the access request complies with the applicable policy, the gateway approves the request, and the Provider Storage System proceeds to deliver the data. If not, access is denied.

This ensures that data access is strictly governed by the token's conditions, offering a secure and policy-driven mechanism for decentralized data sharing.

2.4.2. System Context (Level 2) of TG

The following diagram represents the container level of the C4 model, zooming into the proposed implementation of "Token Gateway" containers. At this level, we focus on the specific responsibilities and functionalities of that system, breaking it down into multiple interacting containers.

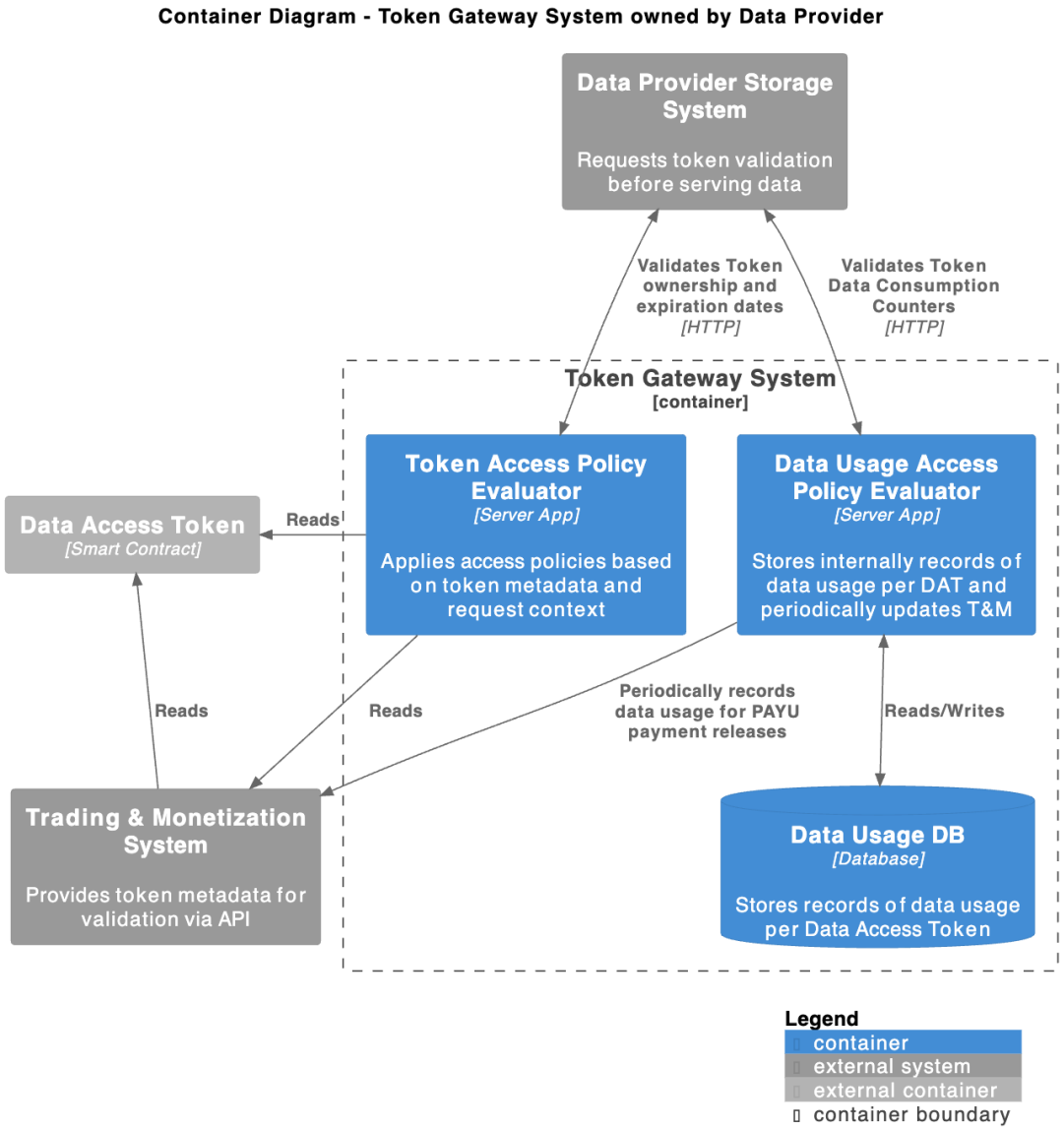

Figure 35 – Token Gateway System C4 Containers Level 2 Diagram

The **Token Gateway System**, implemented, operated and hosted by each Data Provider, is responsible for enforcing access rights granted through Data Access Tokens (DATs) issued by the FAME Trading & Monetization system. While the tokenization and economic logic are handled externally by FAME, the enforcement of those access rules – especially in real-time – is the

responsibility of this system. The Level 2 diagram above presents the internal containers of the Token Gateway System and their interactions with external components.

At the core of the gateway is the **Token Access Policy Evaluator**, a server application responsible for validating whether a request to access data should be permitted. It performs key checks, such as verifying whether the token exists; whether it is owned by the requester; whether it has expired; and whether it corresponds to the correct asset and access policy.

This evaluator fetches token metadata from two optional sources: the FAME Trading & Monetization System (via API) or the underlying smart contract (via read-only smart contract calls). This dual-path validation ensures flexibility between off-chain and on-chain data sources, depending on the provider's chosen trust model.

In the case of **Pay-As-You-Use (PAYU)** models, access cannot be determined by token ownership alone. Instead, it must factor in how much data has already been consumed by the token holder. For this purpose, the **Data Usage Access Policy Evaluator** is introduced. This component keeps track of consumption counters and determines whether a user has remaining access under the current PAYU agreement.

It interfaces directly with the **Data Usage DB**, a local database where counters are stored and updated per Data Access Token. Each time a request is made, this evaluator checks whether the allowed quota has been exceeded before permitting access.

Periodic Reconciliation with Trading & Monetization

To enable token-based billing in PAYU scenarios, the usage counter component periodically synchronizes data usage records with the Trading & Monetization System. This allows the FAME Platform to release escrowed payments to the Data Provider based on actual consumption. The integration maintains trust and automation between decentralized access enforcement and monetization logic.

2.4.3. Containers Components and Code (Level 3 and 4) of TG

The Token Gateway is a system designed to be implemented, deployed and managed independently by each Data Provider. While its core responsibilities and external interfaces are standardized within the FAME architecture – particularly around Data Access Token (DAT) validation – the internal implementation details are intentionally left open.

This design allows Data Providers to implement the Token Gateway according to their own technical environments, security practices, and infrastructure constraints. As a result, Level 3 (component) and Level 4 (code-level) diagrams are not provided within this documentation, since any such representation would only reflect one of many possible implementations.

To support integration and adoption, the FAME consortium will deliver a reference implementation of the Token Gateway and an accompanying Data Storage System, packaged as a demo application. This demo will serve as a practical blueprint for providers wishing to integrate with the FAME Platform quickly and effectively.

All implementation details –including API specifications, architecture decisions, and deployment guidance – will be made available in a dedicated repository, accompanied by technical documentation that can be used as a starting point or adapted to meet provider-specific needs.

3. Modules Installation Guide

3.1 Price Advisory Tool Installation Guide

PAT is implemented in Node.js using Nest framework TypeScript starter repository.

For installation it is necessary to be in the root folder with code.

To install all the necessary packages, run the following in terminal:

```
npm install
```

To start server with swagger, run the following command in terminal:

```
npm run start
```

Then the localhost:7007/swagger page is available with all API services

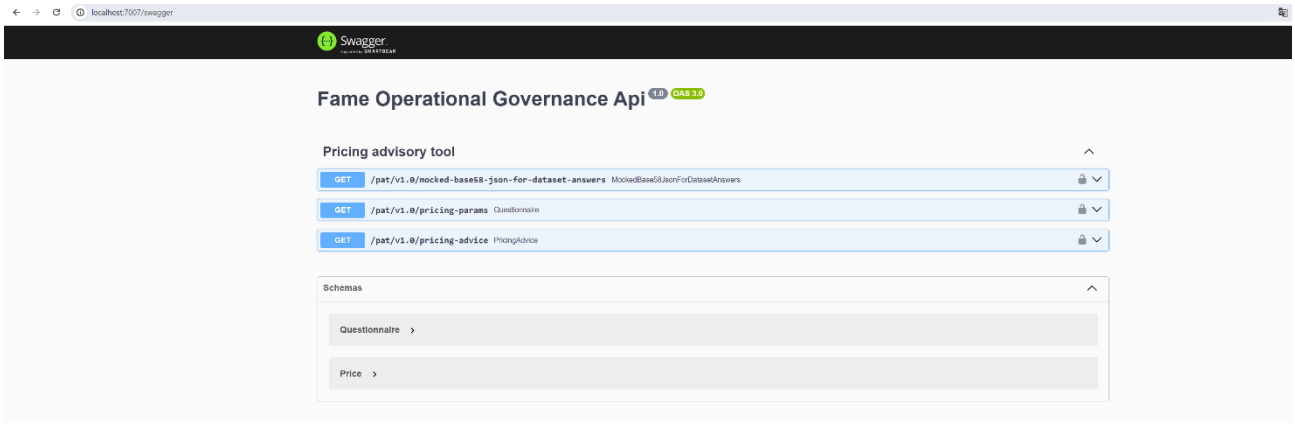

Figure 36 – Price Advisory API Services

3.2 T&M Installation Guide

3.2.1. Overview

Project FAME's Smart Contract Interactions provided by FTS & TRB.

The code repository can be found on [FAME / Framework / tm · GitLab (infinitech-h2020.eu)]

3.2.2. Development Getting Started

Installation

To install all the necessary packages, run the following in terminal:

```
npm install
```

Docker Image Preparation and Publishing

This project uses Docker for containerized deployment. The following npm scripts facilitate building, pushing, and running the Docker images.

Building the Docker Image

To build the Docker image for this project, run:

```
npm run tm-public:docker:build
npm run tm-private:docker:build
```

These commands build the public and private Trading & Monetization applications using the Dockerfile located in the devops directory.

Pushing the Docker Image to a Registry

Before pushing the Docker image, you must be logged in to the Docker registry. To push the built image to the https://harbor.gftinnovation.eu registry, run:

```
npm run tm-public:docker:push
npm run tm-private:docker:push
```

This script will first log you into the `harbor.gftinnovation.eu` registry (you'll be prompted for your credentials), and then push the `harbor.infinitech-h2020.eu/fame/tm:latest` image to the registry.

Latest Docker images will be published here:
https://harbor.gftinnovation.eu/harbor/projects/40/repositories/tm

Running the Docker Image

To run the app locally, use:

```
npm run tm-public:docker:run
npm run tm-private:docker:run
```

Private app runs on port 3000

Public app runs on port 3001

3.2.3. Usage

Deploying Smart Contracts (and upgrading)

To deploy the necessary contracts:

```
npm run contracts:deploy
```

```
npm run contracts:initialize
```

For upgrading diamond-based TradingManagement contracts

```
npx hardhat upgrade-diamond --diamond-name TradingManagement --
facet-name TradingManagementExecutorFacet
```

Running Swagger API

```
npm run tm-private:start
npm run tm-public:start
```

`Swagger API documentation is available at:`

```
http://URL/swagger
```

3.3. Semantic Search Engine Installation Guide

1. Install Node.js on Windows:

- Visit the official [Node.js website] (https://nodejs.org/).
- Download the Windows Installer (.msi) for the latest LTS version.
- Run the downloaded installer, which will guide you through the setup. Default settings should work for most use cases.
- After installation, open the Command Prompt to verify Node.js and npm:

  ```
  >>> node -v
  ```

  ```
  >>> npm -v
  ```

These commands should print the versions of Node.js and npm installed, confirming the installation was successful.

2. Clone the Project Repository:

- Open the Command Prompt and navigate to the directory where you want your project.
- Use the following git command to clone your repository (given URL is an example):

  ```
  >>> git clone https://git-lab.com/Fame_Search.gi
  ```

3. Install Project Dependencies:

- Navigate into your project director:

  ```
  >>> cd your-project
  ```

- Install all dependencies defined in your "**package.json**" by running:

  ```
  >>> npm install
  ```

This command reads the "**package.json**" file and installs all the necessary packages for your project to run. It may take a few minutes depending on the number of dependencies.

4. Serve Your Application:

- Once the dependencies are installed, you can serve your application on a local development server by running:

  ```
  >>> npm start
  ```

- The “start” script in your “**package.json**” is configured to use “**vue-cli-service**” to serve your app. This will compile your Vue application and serve it usually at “**http://localhost:8080**”.

4. Conclusions

In this document, we describe the logic and technical specifications for the second release of three modules contributing to the FAME federated marketplace: the Trading and Monetization (T&M), the Price Advisory (PAT, SAT), and the Semantic Search, all playing an essential role in the Asset Publishing and Offering Definition use cases. The demonstrators, which are the result of activities belonging to T4.2, T4.3, and T4.4, as well as to several other tasks in WP4 and WP3, are presented – from the end user's perspective.

All modules contribute to the overall FAME federated marketplace. However, they can only work when combined with several administrative features – like digital currency management, catalogue editing, etc. These capabilities should be mostly enabled by other platform modules, in particular P&T and Operational Governance (GOV). The consolidation of all work is planned for the second part of the FAME project, and will be finalized in WP2, where the integration and front-end development activities ("Dashboard") take place in.