Federated decentralized trusted dAta Marketplace for Embedded finance

D4.2 - Pricing, Trading and Monetization Techniques I

Title	D4.2 - Pricing, Trading and Monetization Techniques I
Revision Number	1.2
Task reference	T4.2 T4.3 T4.4
Lead Beneficiary	FTS
Responsible	Geert Machtelinckx
Partners	ENG, EUBA, FTS, INNOV, JOT, JRC, TRB, UBI, UIA
Deliverable Type	DEM
Dissemination Level	PU
Due Date	2024-03-31 [Month 15]
Delivered Date	2024-05-23
Internal Reviewers	IDSA INNOV GFT
Quality Assurance	UPRC
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

Revision History

Version	Date	Partners	Description
0.1	2024-02-02	FTS EUBA JOT TRB	Table of Contents
0.2	2024-03-02	FTS EUBA JOT TRB	Integrated version with contributions
0.3	2024-04-02	FTS EUBA JOT TRB	Version for peer review
0.4	2024-05-21	FTS EUBA JOT TRB	Uptate after peer review
0.5	2024-05-22	FTS EUBA JOT TRB	Version for QA
1.0	2024-05-22	FTS EUBA JOT TRB	Reviewed version
1.1	2024-05-22	FTS EUBA JOT TRB	Version with rework
1.2	2024-05-23	FTS EUBA JOT TRB	Version for submission

Definitions

Acronyms	Definition
AAI	authentication authorization infrastructure
AI	Artificial Intelligence
AID	Asset Identifier
AP	Asset Pricing
API	Application Programming Interface
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
JSON	JavaScript Object Notation
MVP	Minimum Viable Product|Platform
NLP	Natural language processing
NLTK	Natural Language Toolkit
PAT	Pricing Advisory Tool
PTM	Pricing, Trading and Monetisation
REST	Representational State Transfer
SAT	Similarity Advisory Tool
SS	Semantic Search
SSE	Semantic Search Engine
TID	Trade Account Identifier
TM	Trade and Monetisation
URL	Uniform Resource Locator

Executive Summary

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

This deliverable is the first tangible result of the combined efforts done under “Pricing, Trading and Monetization Techniques”. It describes the artefacts created in the 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 the form of an integrated demonstrator that is documented in the present document.

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

Task T4.2 is about both a Price Advisory Tool and Similarity Analysis tool, helping data asset owners to set the right pricing to 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 consumer and seller. T4.4 is allowing a user to search for a digital asset, starting from the FDAC’s asset metadata.

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

Key Insights:

- The prototypes are as such not closely interrelated, but all contributed 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 Federated Catalogue of Data Assets, Blockchain-based Data Provenance Infrastructure and Operational Governance, developed in WP4.
- All also relate to work in Platform Architecture, Data Marketplace Platform Integration, Federated AAI Infrastructure, Unified Security Policy Management, and Data Provenance Infrastructure, developed inWP3.

In particular, the document provides information related to:

Modules Overview: Describes the different modules’ role / 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 D4.2 "Pricing, Trading and Monetisation". This is the first of two scheduled releases: it provides the functionalities underpinning the basic FAME Platform in its *minimum viable product* (MVP) configuration. The second release, which is due by M27, will include additional features and will have a tighter integration with the other Platform modules.

In its current form, the D4.2 prototype has been tested and demonstrated in the scope of the MVP Platform.

The output of T4.2 is 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 navigate producers in setting relevant price considering the intrinsic value of digital asset leaving the final price decision on producer solely. PAT contains back-end functionalities for price-advice calculation, and it will include Similarity Analysis Tool, known as SAT for which historical sales have been executed. Current version is based on static pricing mechanism and for next are dynamic (demand-based) as well as skinning and penetration pricing mechanisms planned.

The deliverable of T4.3 (smart contracts for Trade and Monetisation) contains a partially integrated demonstrator has the back-end functionalities available to set up the smart contracts that will allow the creation of an offer and start trading, i.e. exchanging the data asset with a payment token in different payment schemes (over the counter, pay as you go, pay as you use, subscription). The aim is to have this back-end feature used in a user dashboard / front-end.

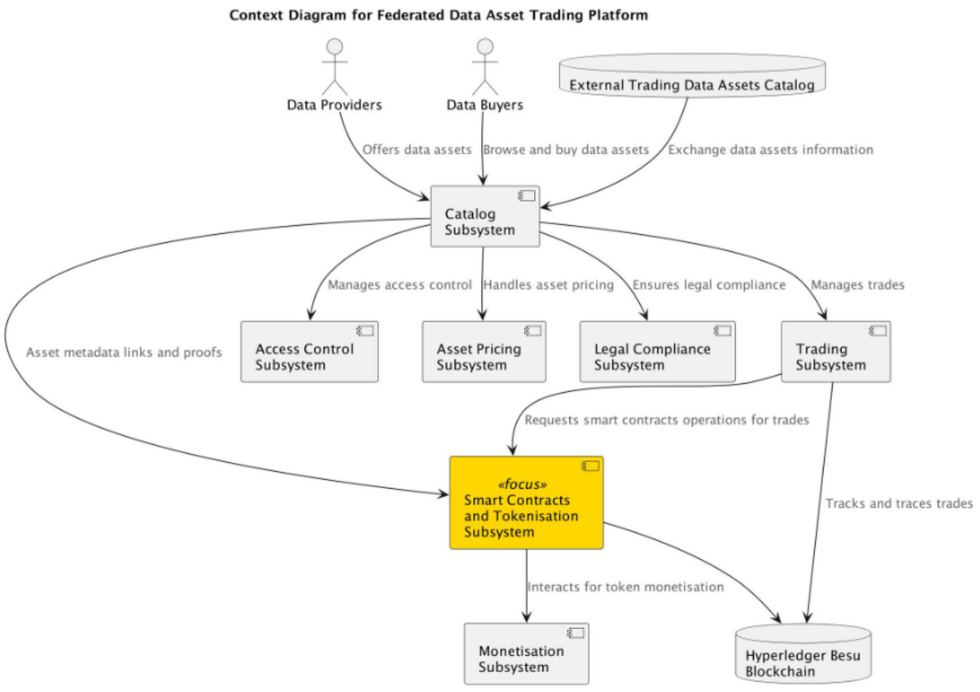

Context Diagram for Federated Data Asset Trading Platform
Data Providers
Data Buyers
External Trading Data Assets Catalog
Offers data assets
Browse and buy data assets
Exchange data assets information
Catalog Subsystem
Manages access control
Handles asset pricing
Ensures legal compliance
Manages trades
Asset metadata links and proofs
Access Control Subsystem
Asset Pricing Subsystem
Legal Compliance Subsystem
Trading Subsystem
Requests smart contracts operations for trades
«focus» Smart Contracts and Tokenisation Subsystem
Tracks and traces trades
Interacts for token monetisation
Monetisation Subsystem
Hyperledger Besu Blockchain

Figure 1 - Context Diagram showing Smart Contracts and Tokenisation Subsystem

The semantic search component (created through task T4.4), focuses on providing an intuitive and effective search functionality within the FAME platform. It processes user queries to deliver 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 approach.

1.2. Insights from other Tasks and Deliverables

Deliverable D4.2 is the result of activities performed in the scope of 3 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’s 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”, 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 will be integrated through T2.3 “Data Marketplace Platform Integration”.

These interdependencies of relationships are depicted in below Figure 2.

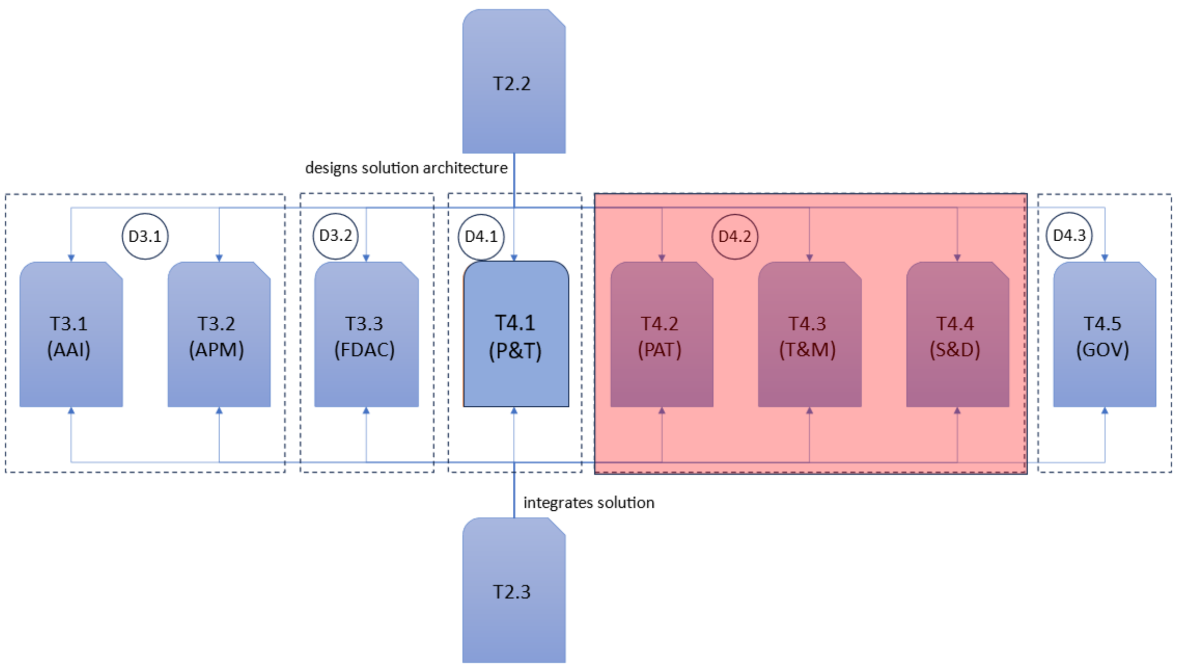


Figure 2 - Tasks / deliverable relationship

1.3. Structure

This document consists of five sections.

1. Introduction, i.e. this introductory section that sets the scene of the deliverable.
2. Module Overview, which sets the context (with reference to the FAME Solution Architecture from deliverable D2.2), provides the general description of the module from the functional perspective, identifies the software Components, and explains their relationship with other modules of the FAME Platform.
3. Components Specification, which 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.
4. Module Demonstration, which gives instructions on how to set up the prototype in a suitable environment and leads the reader through a visual demonstration – represented by screenshots – within a partially integrated FAME Platform.
5. Conclusions, which provides a recap of the main achievements and the outlook for future activities.

2. Modules Overview

The Blockchain-based Pricing, Trading and Monetization (PTM) module of the FAME Platform is responsible for trading activities.

In the C4 architecture model of the FAME Solution Architecture, the three relevant modules are classified as a Container named "Assets Trading & Monetization" (AT&M in brief), "Asset Pricing" (AP in brief) and "Semantic Search" (SS in brief).

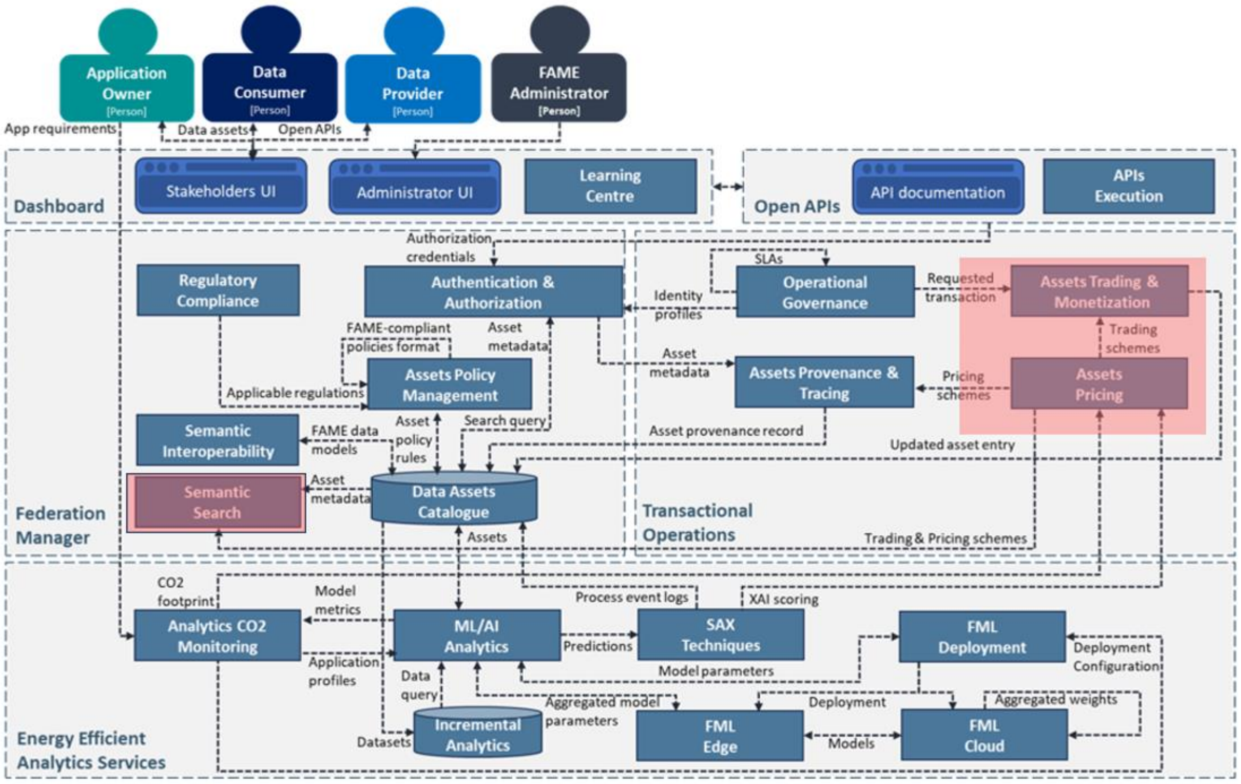


Figure 3 - Modules Overview

2.1. Trading and Monetization

2.1.1. Trading and Monetization C4 Component-level Architecture

The C4 model is a framework for visualizing the architecture of a software system, and it consists of four levels: Context, Container, Component, and Code.

2.1.1.1. Level 1 (Context) Diagram

The high-level overview of the system, its users (or actors), and their interactions are depicted below. The platform comprises seven main subsystems and interacts with two actors and one external database. The system's main purpose is to enable the trading of data assets between the actors "Data Providers" and "Data Consumers."

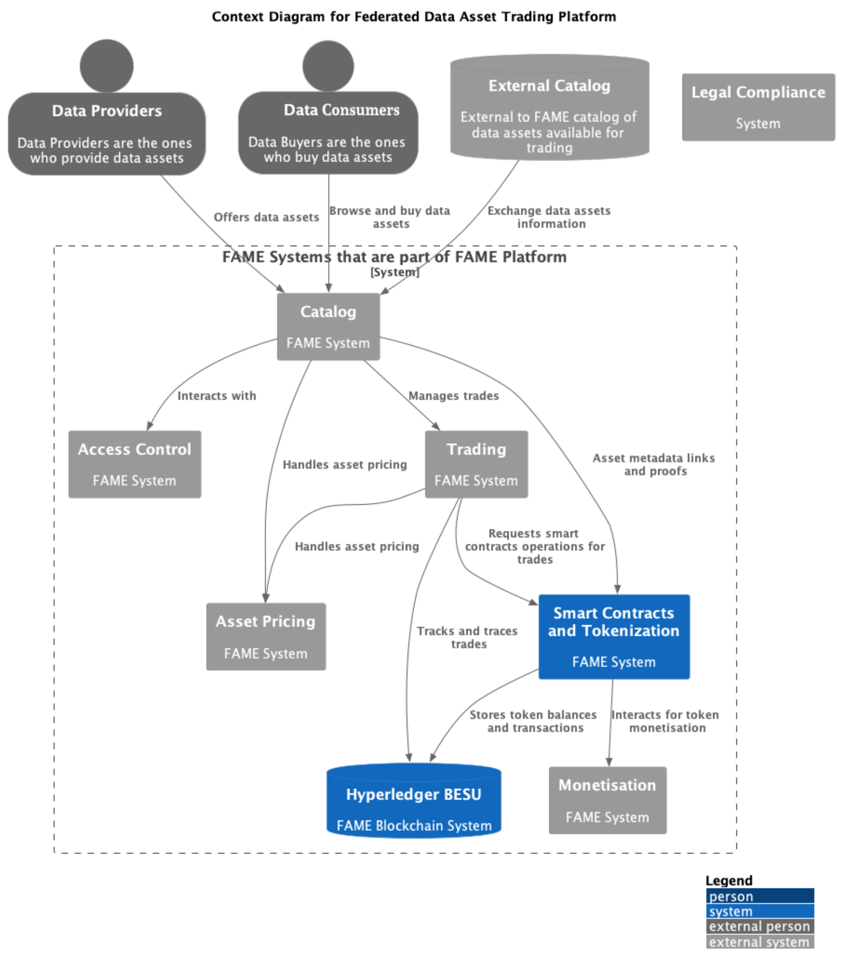

Context Diagram for Federated Data Asset Trading Platform
Data Providers
Data Providers are the ones who provide data assets
Data Consumers
Data Buyers are the ones who buy data assets
External Catalog
External to FAME catalog of data assets available for trading
Legal Compliance
System
Offers data assets
Browse and buy data assets
Exchange data assets information
FAME Systems that are part of FAME Platform
[System]
Catalog
FAME System
Interacts with
Manages trades
Access Control
FAME System
Handles asset pricing
Trading
FAME System
Asset metadata links and proofs
Handles asset pricing
Requests smart contracts operations for trades
Asset Pricing
FAME System
Tracks and traces trades
Smart Contracts and Tokenization
FAME System
Stores token balances and transactions
Interacts for token monetisation
Hyperledger BESU
FAME Blockchain System
Monetisation
FAME System
Legend
person
system
external person
external system

Figure 4 - Context Diagram for Federated Data Asset Trading Platform

Actors:

1. **Data Providers:** These actors offer data assets for trading on the platform.
2. **Data Consumers:** These actors browse and buy the offered data assets.

External Systems:

1. **External Trading Data Assets Catalog:** This external system exchanges data assets information with our system.
2. **Legal Compliance Subsystem:** This subsystem ensures legal compliance for the platform.

Subsystems:

1. **Smart Contracts and Tokenisation Subsystem:** This is the main subsystem we are focusing on. It handles the creation and management of smart contracts and tokens. It interacts with the Catalog Subsystem to receive asset metadata links and proofs, and it interacts with the Monetisation Subsystem for token monetisation.
2. **Access Control Subsystem:** This subsystem manages the access control to the platform.
3. **Asset Pricing Subsystem:** This subsystem handles the pricing of assets.

4. **Catalog Subsystem:** This is the central subsystem that manages data assets offered by Data Providers. It interfaces with all the actors and the majority of other subsystems to handle trades, pricing, access control, legal compliance, and asset metadata.
5. **Monetisation Subsystem:** This subsystem manages the monetisation of tokens in the platform.
6. **Trading Subsystem:** This subsystem manages trades on the platform.
7. **Hyperledger Besu Blockchain**, which is used as a decentralized database to track and trace trades and for interaction with the Smart Contracts and Tokenisation Subsystem.

2.1.1.2. Level 2 (Container) Diagram

The following diagram represents the container level of the C4 model, zooming into the "Smart Contracts and Tokenization Subsystem" of the overall Federated Data Asset Trading Platform. At this level, we focus on the specific responsibilities and functionalities of the subsystem, breaking it down into multiple interacting components.

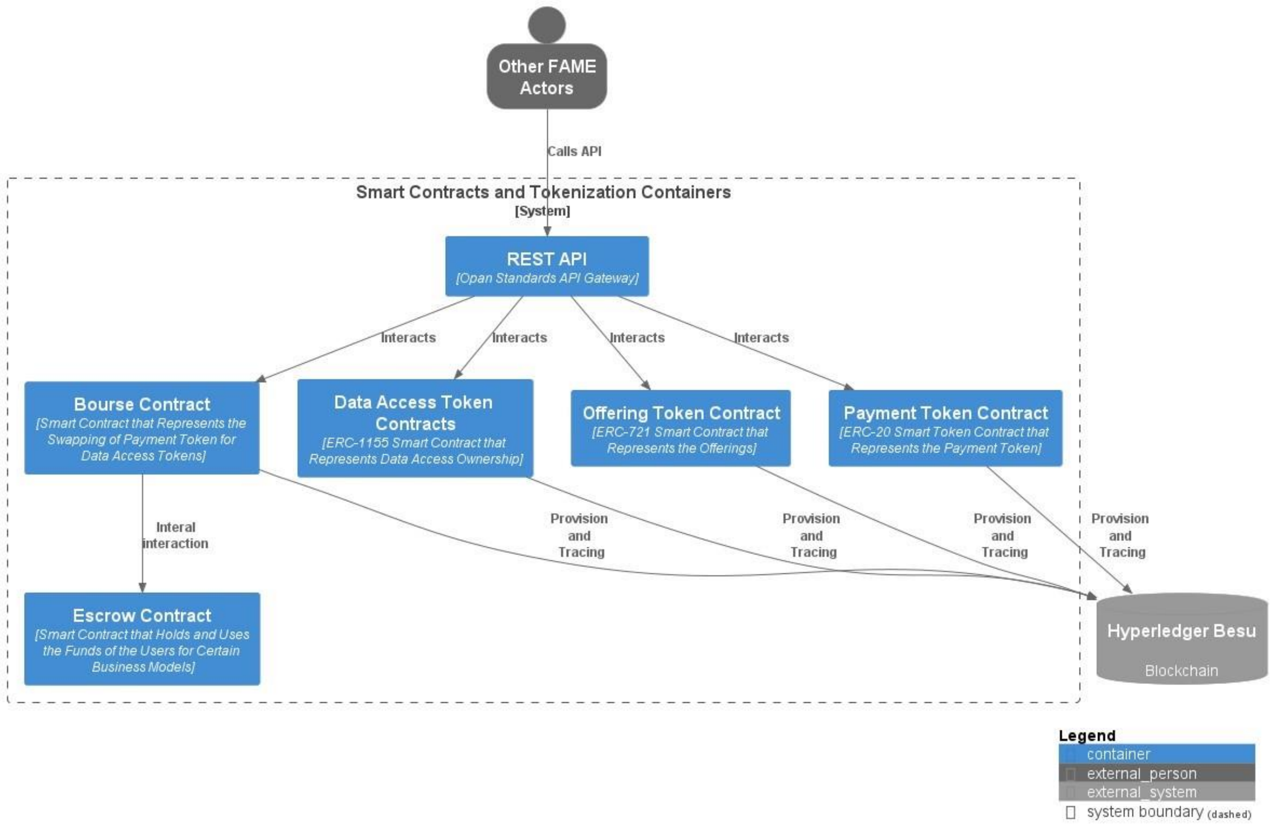

Other FAME Actors
Calls API
Smart Contracts and Tokenization Containers
[System]
REST API
[Opan Standards API Gateway]
Interacts
Interacts
Interacts
Interacts
Bourse Contract
[Smart Contract that Represents the Swapping of Payment Token for Data Access Tokens]
Data Access Token Contracts
[ERC-1155 Smart Contract that Represents Data Access Ownership]
Offering Token Contract
[ERC-721 Smart Contract that Represents the Offerings]
Payment Token Contract
[ERC-20 Smart Token Contract that Represents the Payment Token]
Interal interaction
Provision and Tracing
Provision and Tracing
Provision and Tracing
Provision and Tracing
Escrow Contract
[Smart Contract that Holds and Uses the Funds of the Users for Certain Business Models]
Hyperledger Besu
Blockchain
Legend
container
external_person
external_system
system boundary (dashed)

Figure 5 - Smart Contract & Tokenisation Subsystem Level 2 Diagram

The diagram illustrates the following containers:

1. **REST API**: An open standards API gateway that other FAME actors interact with.
2. **Payment Token Contract**: A token payment system that implements the ERC-20 standard.

3. **Data Access Token Contracts**: Represents the ERC-1155 tokens that provide a representation of data access ownership for different business models.
4. **Offerings Token Contract**: Implements ERC-721 standard where tokens represent the offerings that are added to the catalogue, which makes it available for Data Access Tokens to be purchased.
5. **Bourse Contract**: Smart contract that handles the swapping of currency tokens for data access tokens.
6. **Escrow Contract**: An escrow smart contract that handles holding of Payment Tokens to have a secure and trusted way of depositing/withdrawing for certain business models such as pay-as-you-use.

The **REST API** container interacts with all other containers within the system. Each of these containers are setup in the **Hyperledger Besu** blockchain, which is an external system represented as a database in the diagram. The interactions with the blockchain involve the provision and tracing of transactions.

The **Other FAME Actors** represent external entities that interact with the system by calling the REST API.

2.1.1.3. Level 3 Component Diagram

At this level, the goal is to provide even more detail, breaking down the selected containers into its constituent components.

Open API Specification REST API

Figure 6 provides a detailed view of the **Open API Specification REST API** container, breaking it down into its individual components.

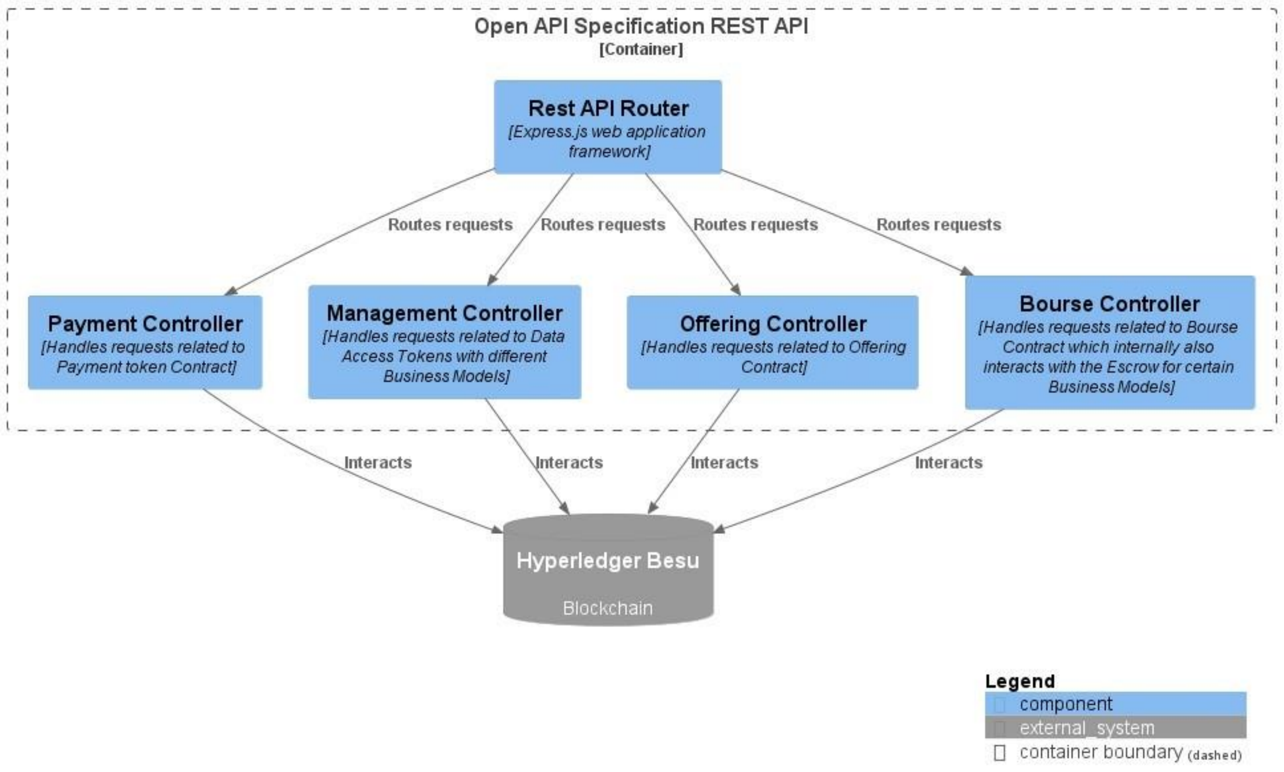

Open API Specification REST API
[Container]
Rest API Router
[Express.js web application framework]
Routes requests
Routes requests
Routes requests
Routes requests
Payment Controller
[Handles requests related to Payment token Contract]
Management Controller
[Handles requests related to Data Access Tokens with different Business Models]
Offering Controller
[Handles requests related to Offering Contract]
Bourse Controller
[Handles requests related to Bourse Contract which internally also interacts with the Escrow for certain Business Models]
Interacts
Interacts
Interacts
Interacts
Hyperledger Besu
Blockchain
Legend
component
external_system
container boundary (dashed)

Figure 6 - Open API Specification REST API for T&M

Figure 6 includes the following components:

1. **REST API Router**: This is the main component that handles routing of requests. It's implemented using the Express.js with Nestjs wrapper web application framework.
2. **Payment Controller**: This component handles requests related to the Payment Token Contract.
3. **Data Access Controller**: This component handles requests related to all Data Access Token Contracts.
4. **Offering Controller**: This component handles requests related to the Offering Token Contract.
5. **Bourse Controller**: This component handles requests related to Bourse Contract and Escrow.

The **REST API Router** routes requests to the appropriate controller based on the request's details. Each of these controllers 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 controller.

ERC-1155 Data Access Token Smart Contracts

Figure 7 provides a detailed view of the **ERC-1155 Data Access Token Smart Contract** container, breaking it down into its individual components.

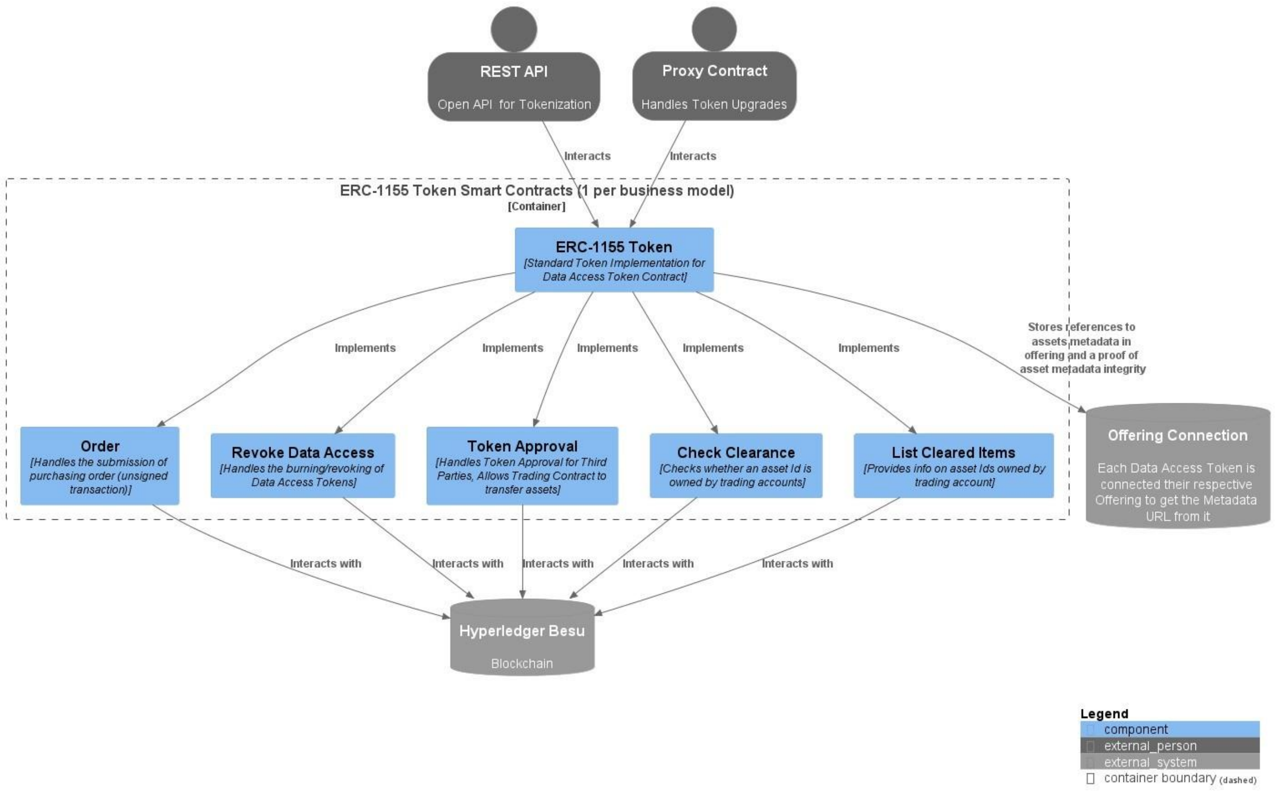

REST API
Open API for Tokenization
Proxy Contract
Handles Token Upgrades
Interacts
Interacts
ERC-1155 Token Smart Contracts (1 per business model)
[Container]
ERC-1155 Token
[Standard Token Implementation for Data Access Token Contract]
Implements
Stores references to assets metadata in offering and a proof of asset metadata integrity
Order
[Handles the submission of purchasing order (unsigned transaction)]
Revoke Data Access
[Handles the burning/revoking of Data Access Tokens]
Token Approval
[Handles Token Approval for Third Parties, Allows Trading Contract to transfer assets]
Check Clearance
[Checks whether an asset Id is owned by trading accounts]
List Cleared Items
[Provides info on asset Ids owned by trading account]
Offering Connection
Each Data Access Token is connected their respective Offering to get the Metadata URL from it
Interacts with
Hyperledger Besu
Blockchain
Legend
component
external_person
external_system
container boundary (dashed)

Figure 7 - ERC-1155 Data Access Token Smart Contract

The diagram includes the following components:

1. **ERC-1155 Token**: This is the main component that implements the standard ERC-1155 token for data access.
2. **Submissions/Order**: This component handles the purchasing of data access tokens.
3. **Revoke Data Access**: This component handles the destruction of data access tokens (based on some requirements).
4. **Token Approval**: This component handles token approval for third parties. It allows the Trading Contract to transfer assets.
5. **Check Clearance**: This component provides information on whether the passed asset id is owned by the passed trading accounts.
6. **List Cleared Items**: This component provides information on which asset ids the passed trading account owns.

The **REST API** and the **Proxy Contracts** interact with their respective **ERC-1155 Data Access Token Contract** component, which in turn implements the functionality of the other components. 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.

The **ERC-1155 Data Access Token Contract** component also interacts with the **Asset Metadata** storage, storing references to assets metadata and a proof of asset metadata integrity.

ERC-721 Offering Smart Contract

Figure 8 provides a detailed view of the **ERC-721 Offering Smart Contract** container, breaking it down into its individual components.

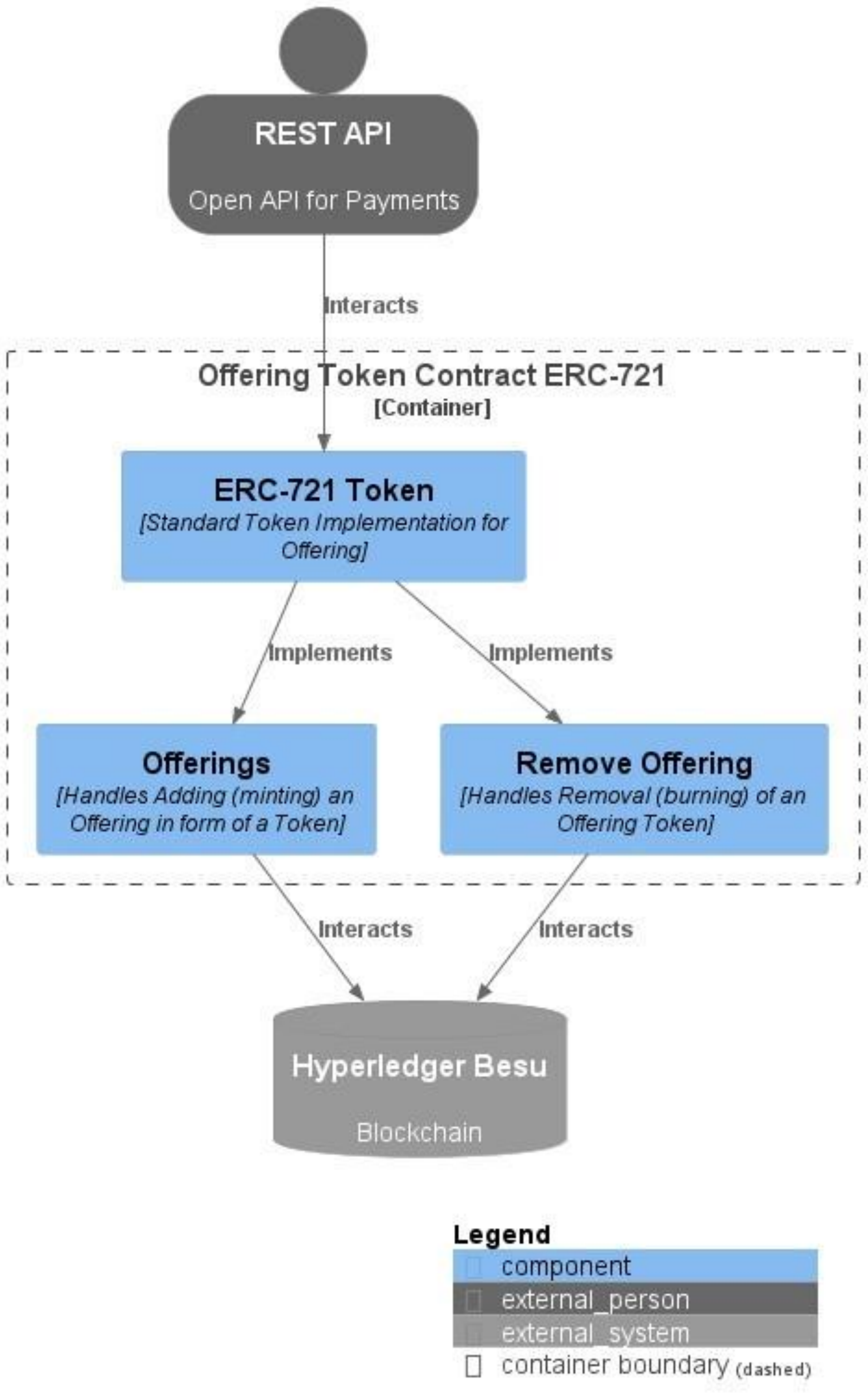

REST API
Open API for Payments
Interacts
Offering Token Contract ERC-721
[Container]
ERC-721 Token
[Standard Token Implementation for Offering]
Implements
Implements
Offerings
[Handles Adding (minting) an Offering in form of a Token]
Remove Offering
[Handles Removal (burning) of an Offering Token]
Interacts
Interacts
Hyperledger Besu
Blockchain
Legend
component
external_person
external_system
container boundary (dashed)

Figure 8 - ERC-721 Offering Token Contract

The diagram includes the following components:

1. **ERC-721 Offering Token**: This is the main component that implements the standard ERC-721 token for offerings.
2. **Offerings**: This component initializes a blockchain trading environment for a given offering by minting a ERC-721 token for it.
3. **Remove Offering**: This component handles the destruction of an offering id.

The **REST API** interact with the **ERC-721 Offering Token Contract** component, which in turn implements the functionality of the other components. 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.

Bourse Contract

Figure 8 provides a detailed view of the **Bourse Contract** container, breaking it down into its individual components.

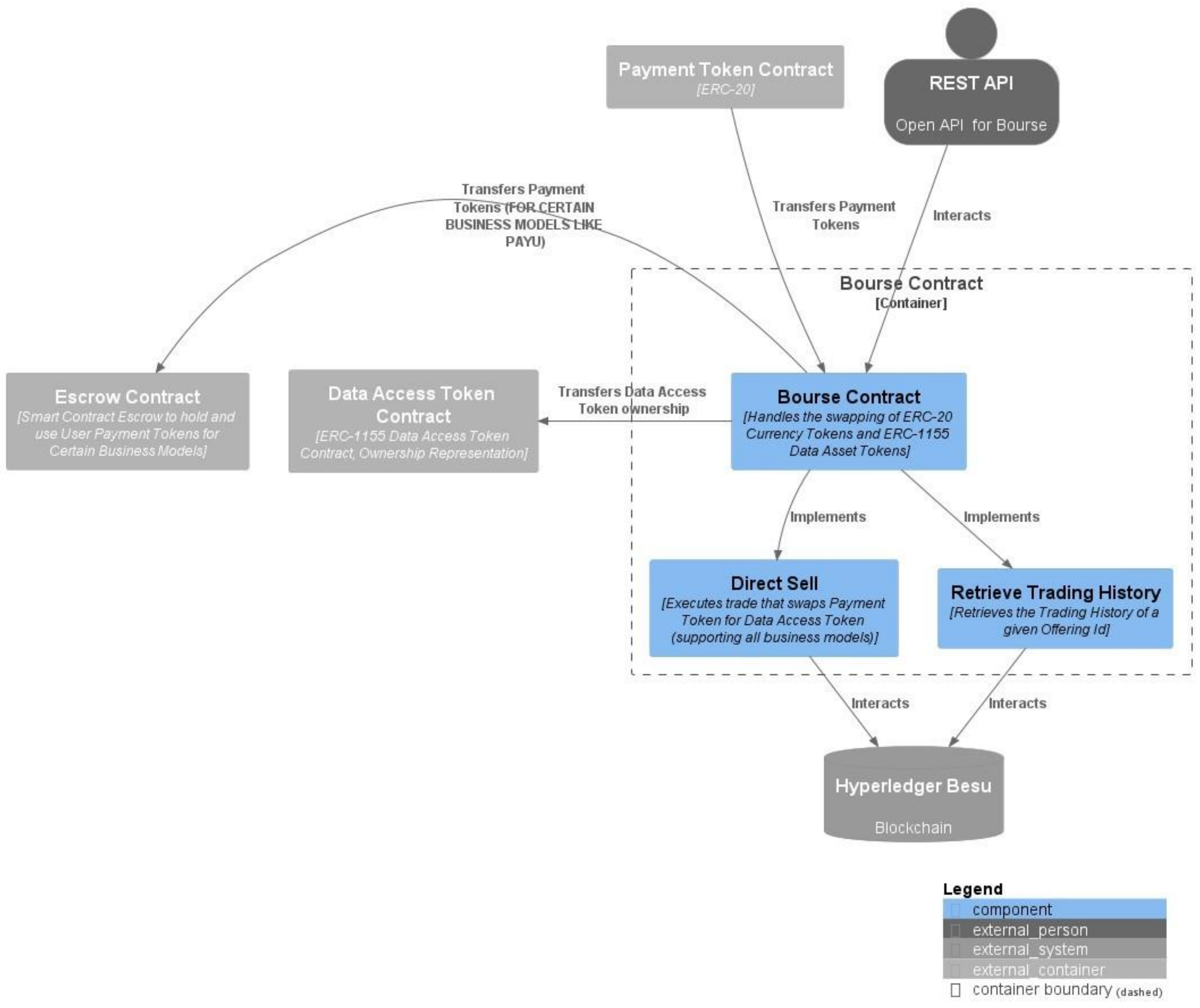

Payment Token Contract
[ERC-20]
REST API
Open API for Bourse
Transfers Payment Tokens (FOR CERTAIN BUSINESS MODELS LIKE PAYU)
Transfers Payment Tokens
Interacts
Bourse Contract
[Container]
Escrow Contract
[Smart Contract Escrow to hold and use User Payment Tokens for Certain Business Models]
Data Access Token Contract
[ERC-1155 Data Access Token Contract, Ownership Representation]
Transfers Data Access Token ownership
Bourse Contract
[Handles the swapping of ERC-20 Currency Tokens and ERC-1155 Data Asset Tokens]
Implements
Implements
Direct Sell
[Executes trade that swaps Payment Token for Data Access Token (supporting all business models)]
Retrieve Trading History
[Retrieves the Trading History of a given Offering Id]
Interacts
Interacts
Hyperledger Besu
Blockchain
Legend
component
external_person
external_system
external_container
container boundary (dashed)

Figure 9 - Bourse Contract

The diagram includes the following components:

1. **Bourse Contract**: This is the main component that handles the swapping of ERC-20 Payment Tokens and ERC-1155 Data Access Tokens.
2. **Direct Sell**: This component executes trades based on contract terms while validating the terms of a trade.
3. **Retrieve Trading History**: This component provides information on the history of a passed offering id based on the previous transactions.

The **REST API** interacts with the **Bourse Contract** component, which in turn implements the functionality of the **Direct Sell** and **Retrieve Trading History** components. The Direct Sell is called automatically when called **Submissions/Order** 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.

The **Bourse Contract** component also interacts with the **Payment Token, Data Access Token and Escrow Contract** containers, handling the payment and transfer of token ownership respectively.

In addition to the above-mentioned functionalities, Bourse contract is also setup for providing future support for first come first served bidding in demand driven auction system.

Escrow Contract

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

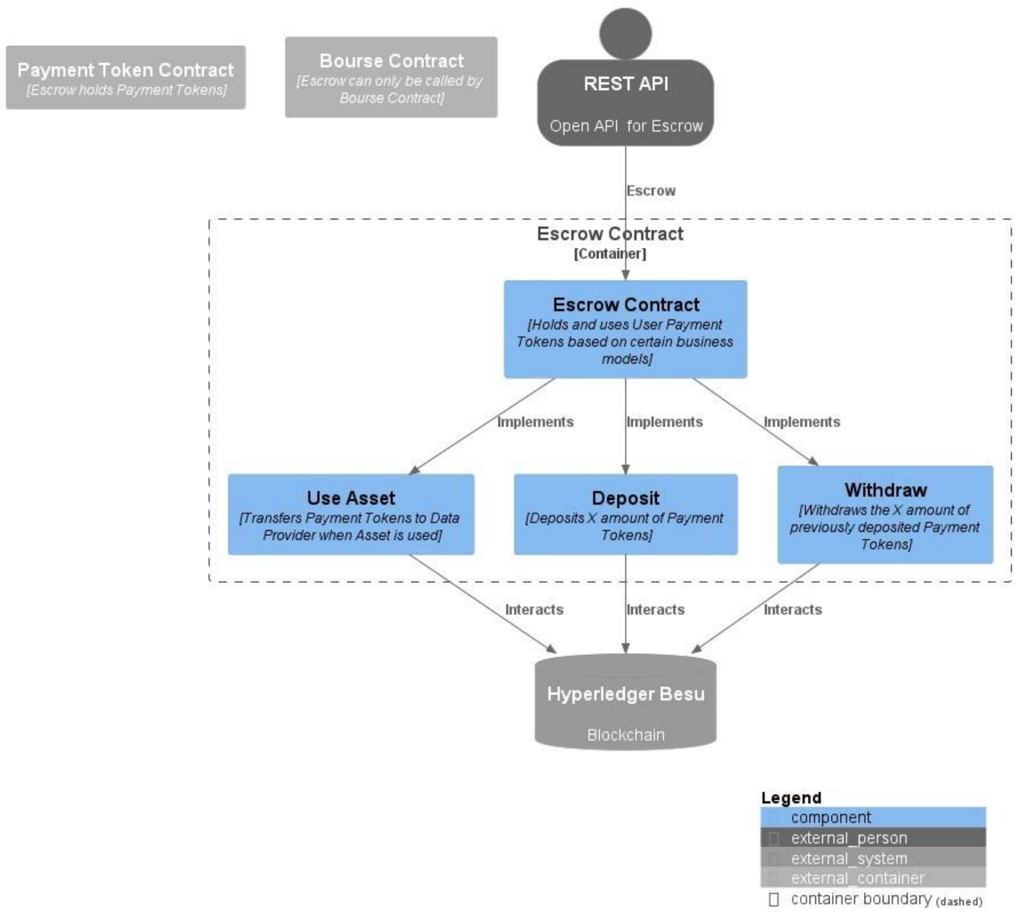

Payment Token Contract
[Escrow holds Payment Tokens]
Bourse Contract
[Escrow can only be called by Bourse Contract]
REST API
Open API for Escrow
Escrow
Escrow Contract
[Container]
Escrow Contract
[Holds and uses User Payment Tokens based on certain business models]
Implements
Implements
Implements
Use Asset
[Transfers Payment Tokens to Data Provider when Asset is used]
Deposit
[Deposits X amount of Payment Tokens]
Withdraw
[Withdraws the X amount of previously deposited Payment Tokens]
Interacts
Interacts
Interacts
Hyperledger Besu
Blockchain
Legend
component
external_person
external_system
external_container
container boundary (dashed)

Figure 10 - Escrow Contract

The diagram includes the following components:

1. **Escrow Contract**: This is the main component that handles the holding and using the ERC-20 Payment Tokens.
2. **Use Asset**: This component uses X amount of Payment Tokens based on the used amount of services by the user and transfers those tokens to the Beneficiary.
3. **Deposit**: This component is used to deposit X amount of Payment Tokens to the Escrow when purchasing access token.
4. **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.

2.1.2. Components and APIs for 'Trade and Monetisation'

This document provides an overview of the Trading & Monetisation (TM) module for the FAME project, detailing the API endpoints available for developers. It is automatically generated from the API definitions provided in the codebase. Swagger scans the annotations and comments in the API code and generates a human-readable documentation that includes all available endpoints, parameters, response formats, and example requests and responses.

Documentation of these APIs can be found in Annex 1 of this document.

2.1.2.1. Setup trading

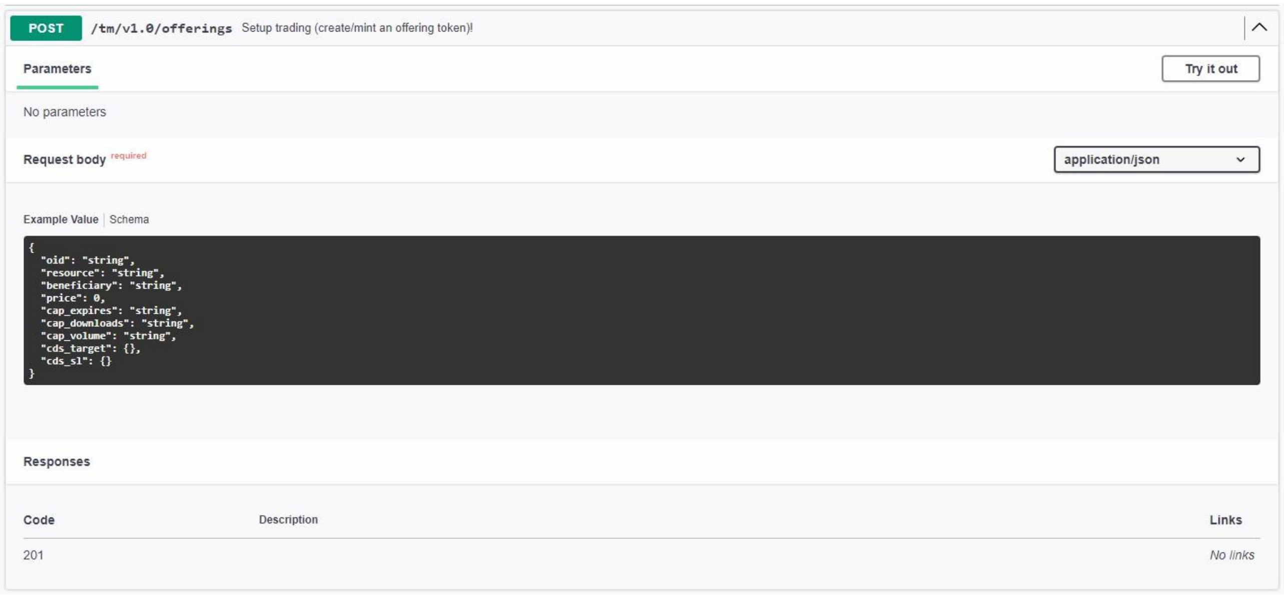
Description

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.



`/tm/v1.0/offerings` Setup trading (create/mint an offering token)!

No parameters

Request body required: application/json

```
{
  "oid": "string",
  "resource": "string",
  "beneficiary": "string",
  "price": 0,
  "cap_expires": "string",
  "cap_downloads": "string",
  "cap_volume": "string",
  "cds_target": {},
  "cds_sl": {}
}
```

Responses

Code	Description	Links
201		No links

Figure 11 - Set up Trading API

2.1.2.2. Submit Purchasing Order

Description

Prepares and returns an unsigned transaction of a purchasing order for the interacting user, as illustrated in Figure 12.

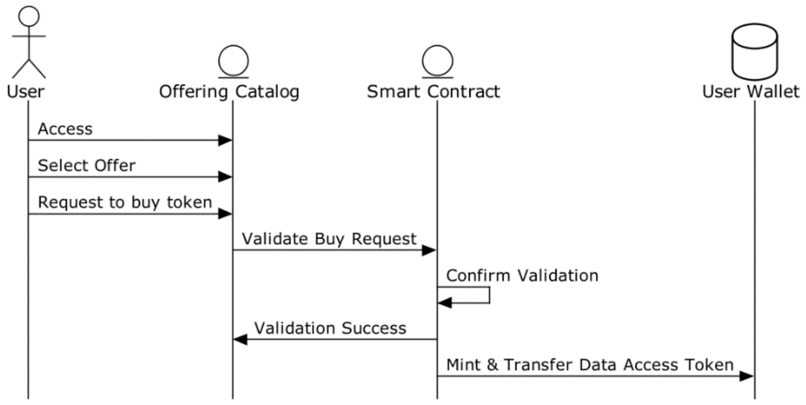
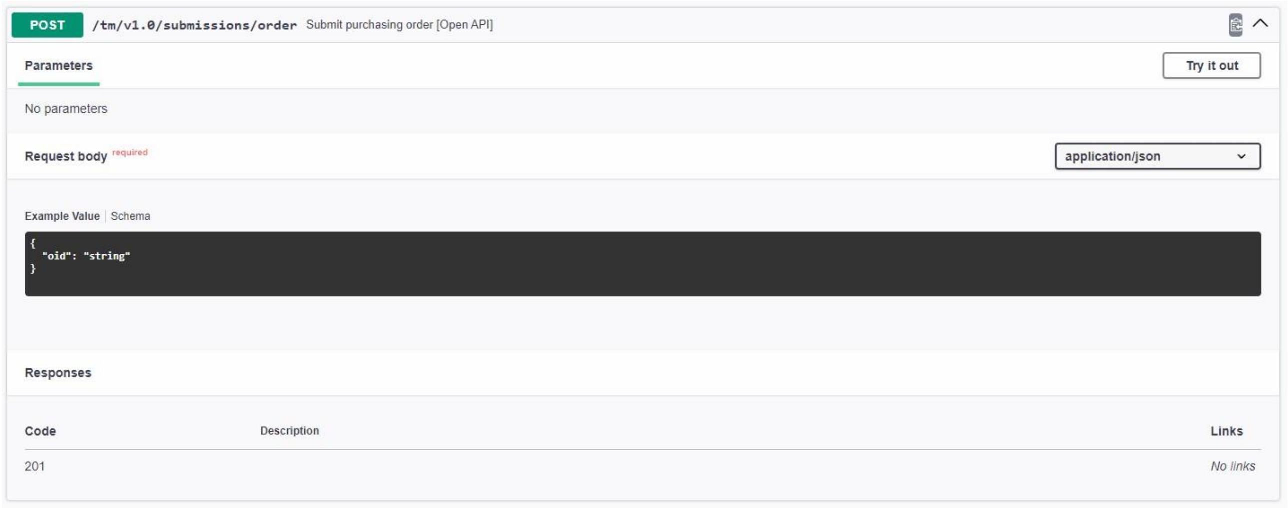


Figure 12 - Submit Purchasing Order Sequence Diagram

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.


POST /tm/v1.0/submissions/order Submit purchasing order [Open API]

Parameters

No parameters

Request body required — application/json

Example Value | Schema

```
{
  "oid": "string"
}
```

Responses

Code	Description	Links
201		No links

Figure 13 - Submit Purchasing Order API

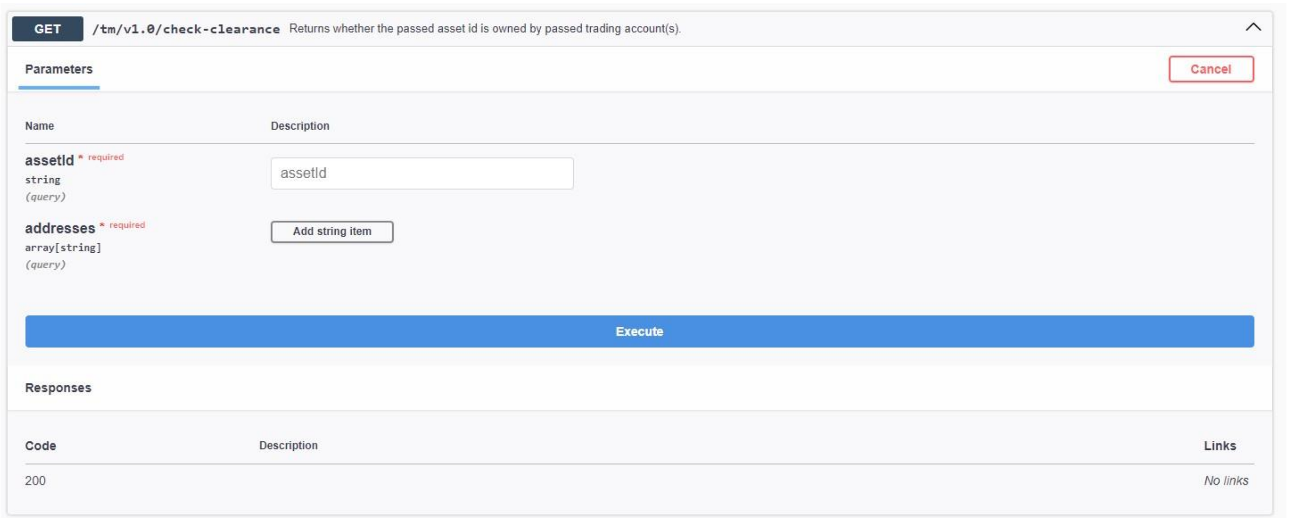
2.1.2.3. Check Clearance

Description

Returns whether one or more trading accounts own a given AID.

Technical Specification

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


GET /tm/v1.0/check-clearance Returns whether the passed asset id is owned by passed trading account(s).

Parameters

Cancel

Name	Description
assetId * required string (query)	assetId
addresses * required array[string] (query)	Add string item

Execute

Responses

Code	Description	Links
200		No links

Figure 14 - Check Clearance API

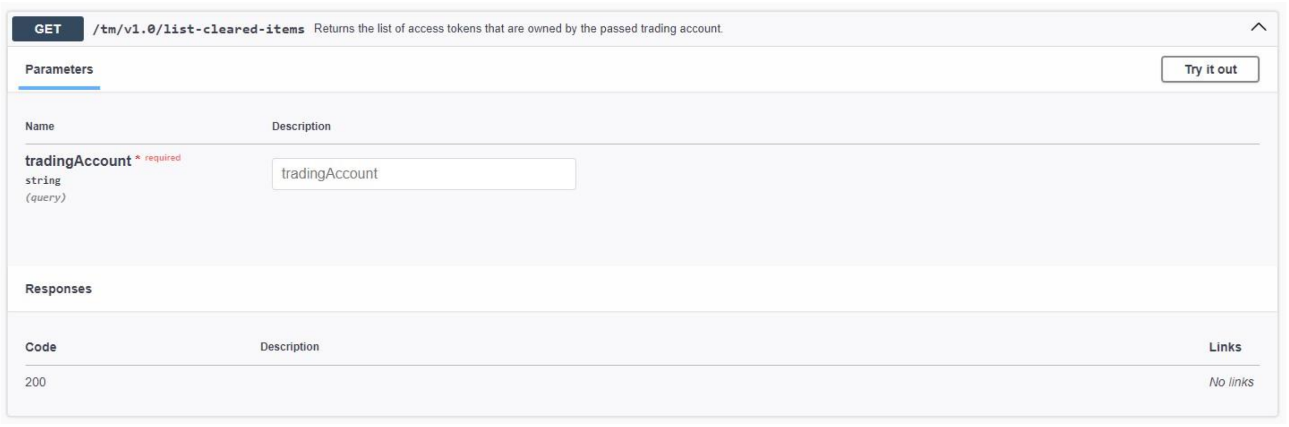
2.1.2.4. List Cleared Items

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.


GET /tm/v1.0/list-cleared-items Returns the list of access tokens that are owned by the passed trading account.

Parameters

Try it out

Name	Description
tradingAccount * required string (query)	tradingAccount

Responses

Code	Description	Links
200		No links

Figure 15 - List Cleared Items API

2.1.2.5. 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.

Technical Specification

GET tm/v1.0/retrieve-trading-history/getNumberOfTradePairs
GET tm/v1.0/retrieve-trading-history/getIthTradePair/ithp
GET tm/v1.0/retrieve-trading-history/getTotalNumVolBuys/ta1/ti1/ta2/ti2
GET tm/v1.0/retrieve-trading-history/getTotalNumVolSells/ta1/ti1/ta2/ti2
GET tm/v1.0/retrieve-trading-history/getNumberOfTraderBuys/tid/ta1/ti1/ta2/ti2
GET tm/v1.0/retrieve-trading-history/getNumberOfTraderSells/tid/ta1/ti1/ta2/ti2
GET tm/v1.0/retrieve-trading-history/getIthTraderSell/ith/tid/ta1/ti1/ta2/ti2
GET tm/v1.0/retrieve-trading-history/getIthTraderBuy/ith/tid/ta1/ti1/ta2/ti2
GET tm/v1.0/retrieve-trading-history/getNumberOfSells/ta1/ti1/ta2/ti2
GET tm/v1.0/retrieve-trading-history/getNumberOfBuys/ta1/ti1/ta2/ti2
GET tm/v1.0/retrieve-trading-history/getIthBuy/ith/ta1/ti1/ta2/ti2
GET tm/v1.0/retrieve-trading-history/getIthSell/ith/ta1/ti1/ta2/ti2
Parameters ithp : ith trade pair (data token, currency coin) ta1 : first token address in trade pair ti2 : first token index in trade pair ta2 : second token address in trade pair ti2 : second token index in trade pair ith : ith buy or sell transaction

Table 1 - Retrieve Trading History 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. In its terminal phase, the strategy intends to incorporate interest-based metrics derived from the quantification of user interactions (clicks) with analogous assets on the FDAC platform. Nonetheless, the determination of asset pricing within 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 16 presents an overview of the price advising process, its actors, and their interactions.

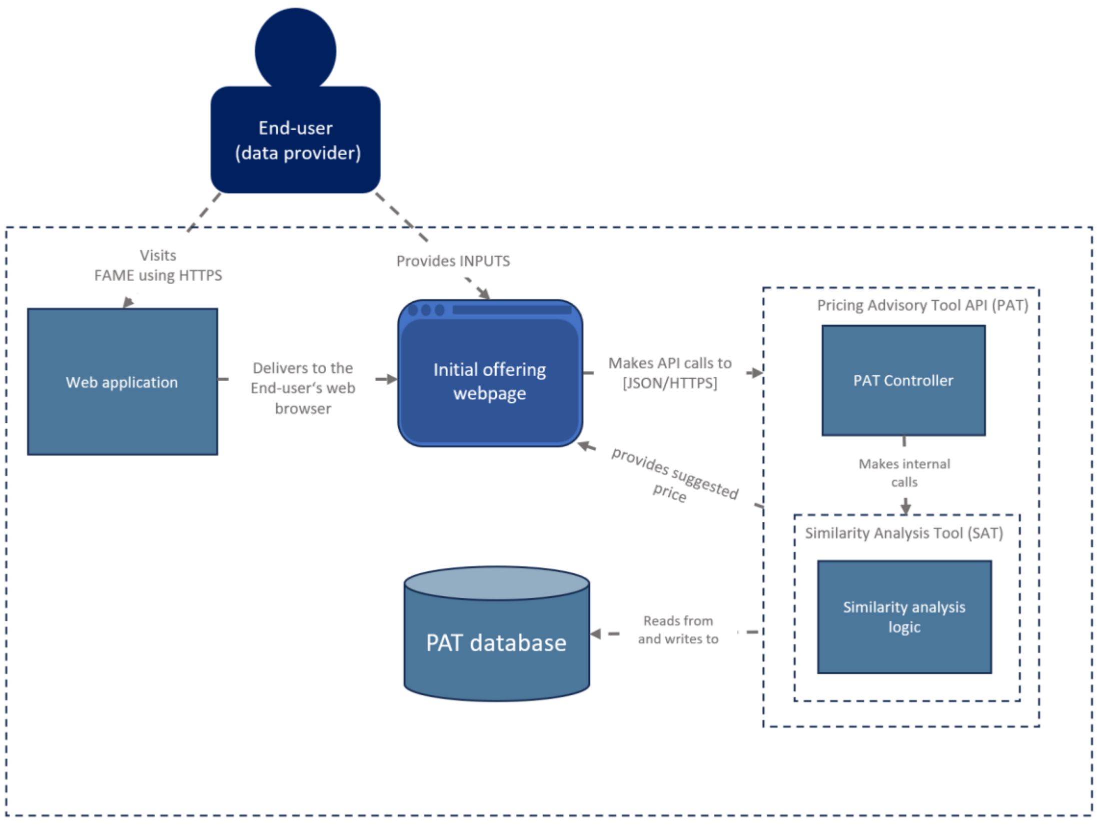


Figure 16 - Price Advising C4 Component-level Architecture

2.2.2. API for Pricing Advisory Tool (PAT)

Description

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 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 17).

Technical Specification

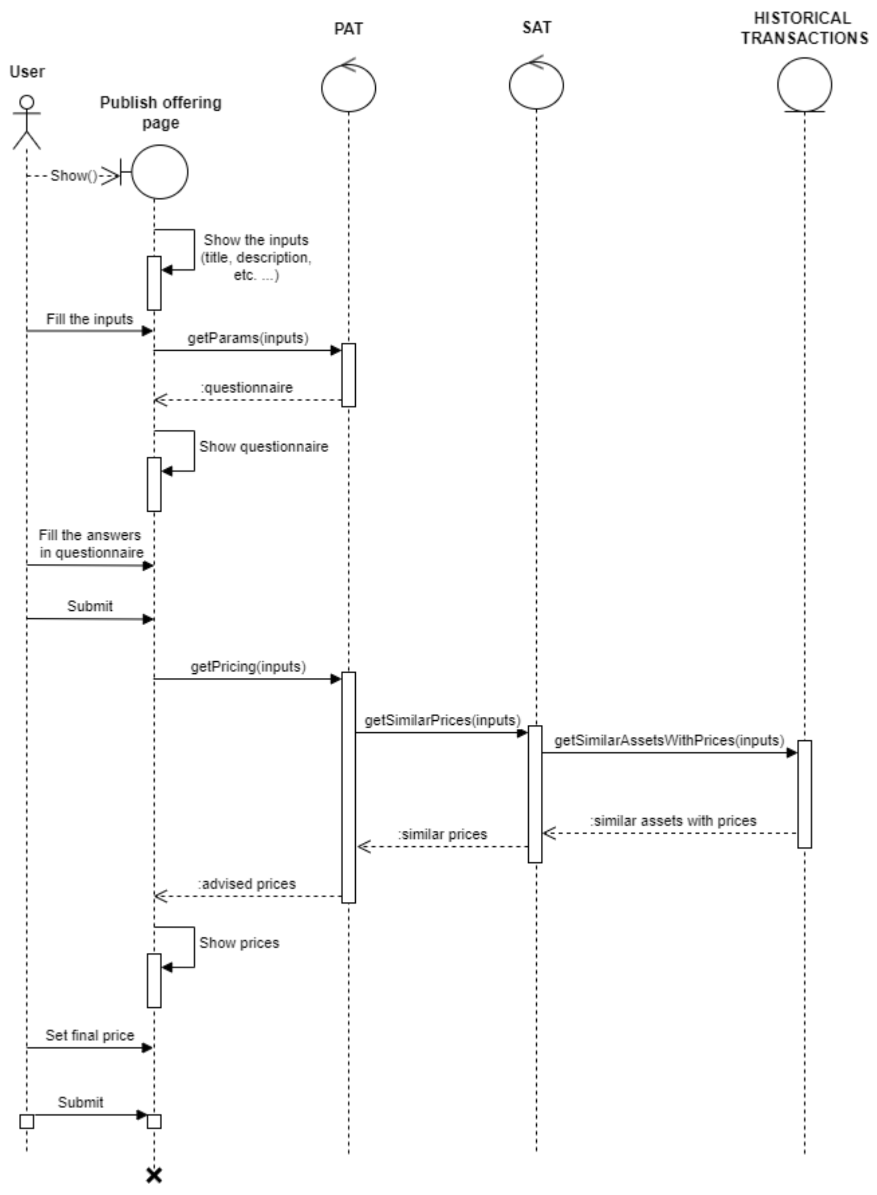
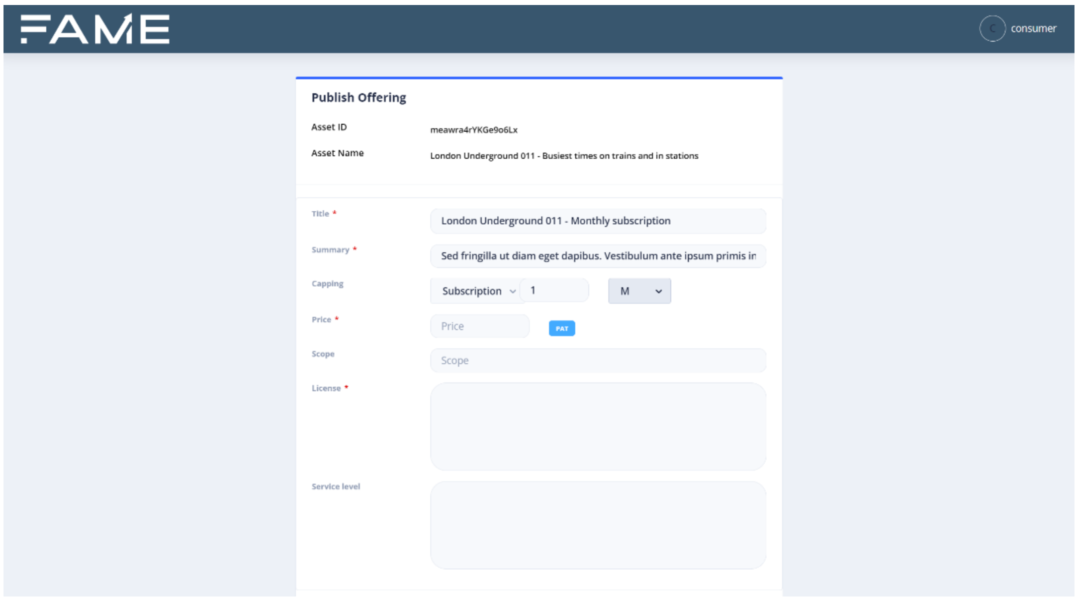


Figure 17 - Price Advisory Sequence Diagram

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.


Publish Offering

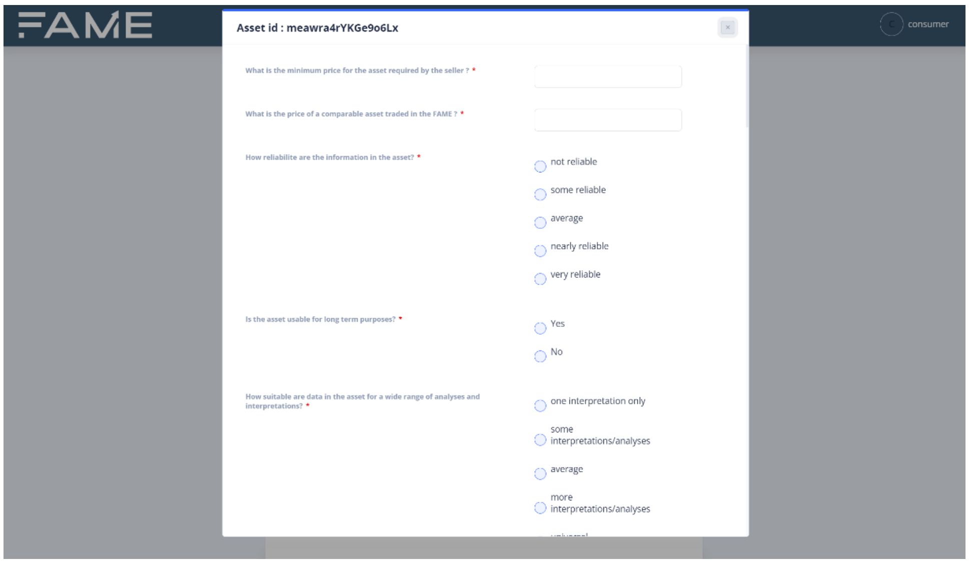
Asset ID	meawra4rYKGe9o6Lx
Asset Name	London Underground 011 - Busiest times on trains and in stations
Title *	London Underground 011 - Monthly subscription
Summary *	Sed fringilla ut diam eget dapibus. Vestibulum ante ipsum primis in
Capping	Subscription 1 M
Price *	Price
Scope	Scope
License *	
Service level	

Figure 18 - 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 step is contingent on the existence of preceding assets.
4. 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.
5. 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 frontend, where they are displayed to the end-user.


Asset id : meawra4rYKGe9o6Lx

What is the minimum price for the asset required by the seller ? *

What is the price of a comparable asset traded in the FAME ? *

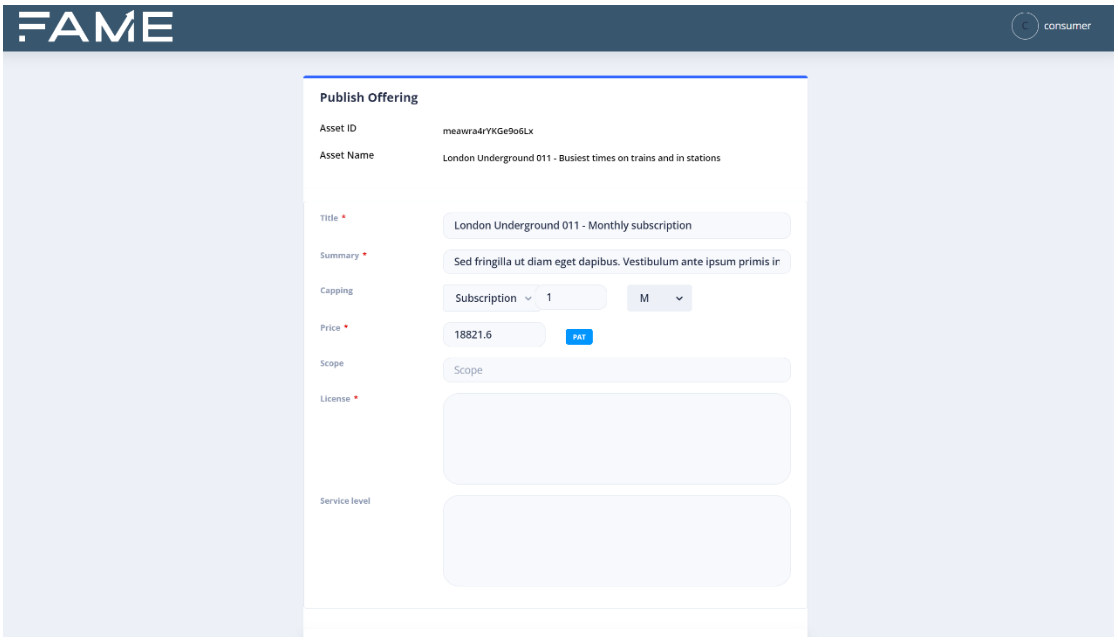
How reliabilite are the information in the asset? *
- not reliable
- some reliable
- average
- nearly reliable
- very reliable

Is the asset usable for long term purposes? *
- Yes
- No

How suitable are data in the asset for a wide range of analyses and interpretations? *
- one interpretation only
- some interpretations/analyses
- average
- more interpretations/analyses

Figure 19 - 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 frontend. The user then has the option to utilize the suggested price or enter their own.



Publish Offering

Asset ID meawra4rYKGe9o6Lx

Asset Name London Underground 011 - Busiest times on trains and in stations

Title * London Underground 011 - Monthly subscription

Summary * Sed fringilla ut diam eget dapibus. Vestibulum ante ipsum primis ir

Capping Subscription 1 M

Price * 18821.6 PAT

Scope Scope

License *

Service level

Figure 20 - Price Recommendation Interface

Table 2 - 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 demonstrator in MVP phase, we use PAT as a tool to navigate business offering creator in setting price by calculating static intrinsic value of the asset. For next version of D4.2 we will use dynamic pricing mechanisms, such as skimming and penetration pricing as well as demand-based pricing used in auctions for digital assets.

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 subsets 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 21 - SAT formulation

where l, m, and n are numbers of logical, continuos 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 no 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 a 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

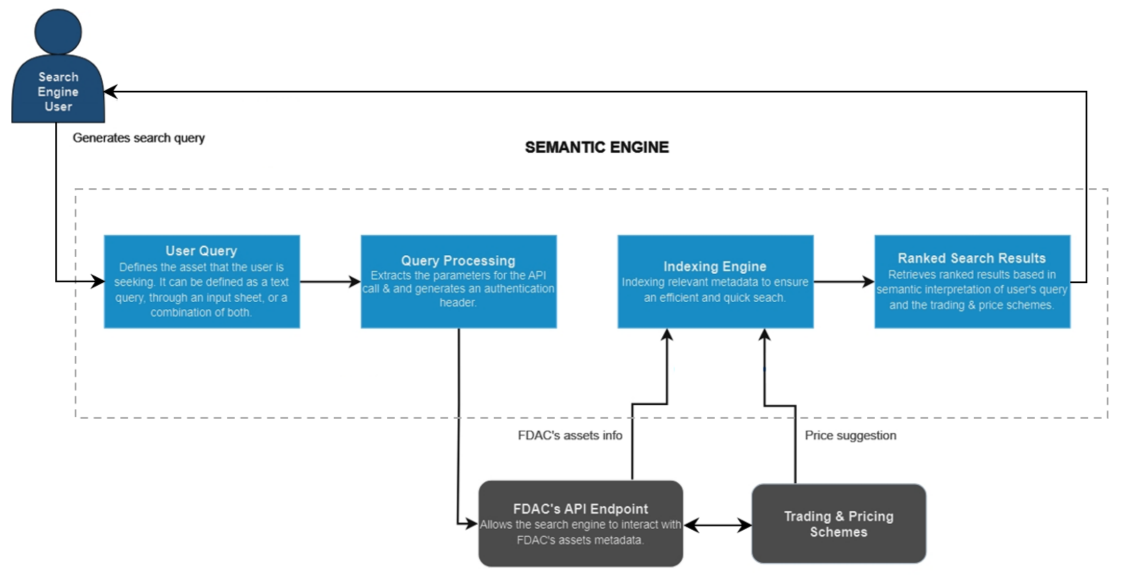

Search Engine User
Generates search query
SEMANTIC ENGINE
User Query
Defines the asset that the user is seeking. It can be defined as a text query, through an input sheet, or a combination of both.
Query Processing
Extracts the parameters for the API call & and generates an authentication header.
Indexing Engine
Indexing relevant metadata to ensure an efficient and quick seach.
Ranked Search Results
Retrieves ranked results based in semantic interpretation of user's query and the trading & price schemes.
FDAC's assets info
Price suggestion
FDAC's API Endpoint
Allows the search engine to interact with FDAC's assets metadata.
Trading & Pricing Schemes

Figure 22 - 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.

2. **Query Processing**: In this stage, the search engine processes the user's input. It extracts the search terms and any additional parameters necessary for the FDAC API call and generates an authentication header to ensure secure API communication.
3. **Indexing Engine**: Here, the engine interacts with the FDAC's asset metadata. It indexes important information to enable efficient and quick searches.
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. The results are ordered based on relevance to the query and potentially by the trading and pricing schemes when integrated into the ranking logic.

2.3.3. External components

1. **FDAC's API Endpoint**: This component represents the interface through which the search engine communicates with the FDAC. It sends requests to this endpoint to retrieve asset metadata that matches the user's search criteria.
2. **Trading & Pricing Schemes**: This component influences the outcome by providing additional data that can affect the ranking of search results, such as price suggestions based on the asset's trading and pricing schemes.

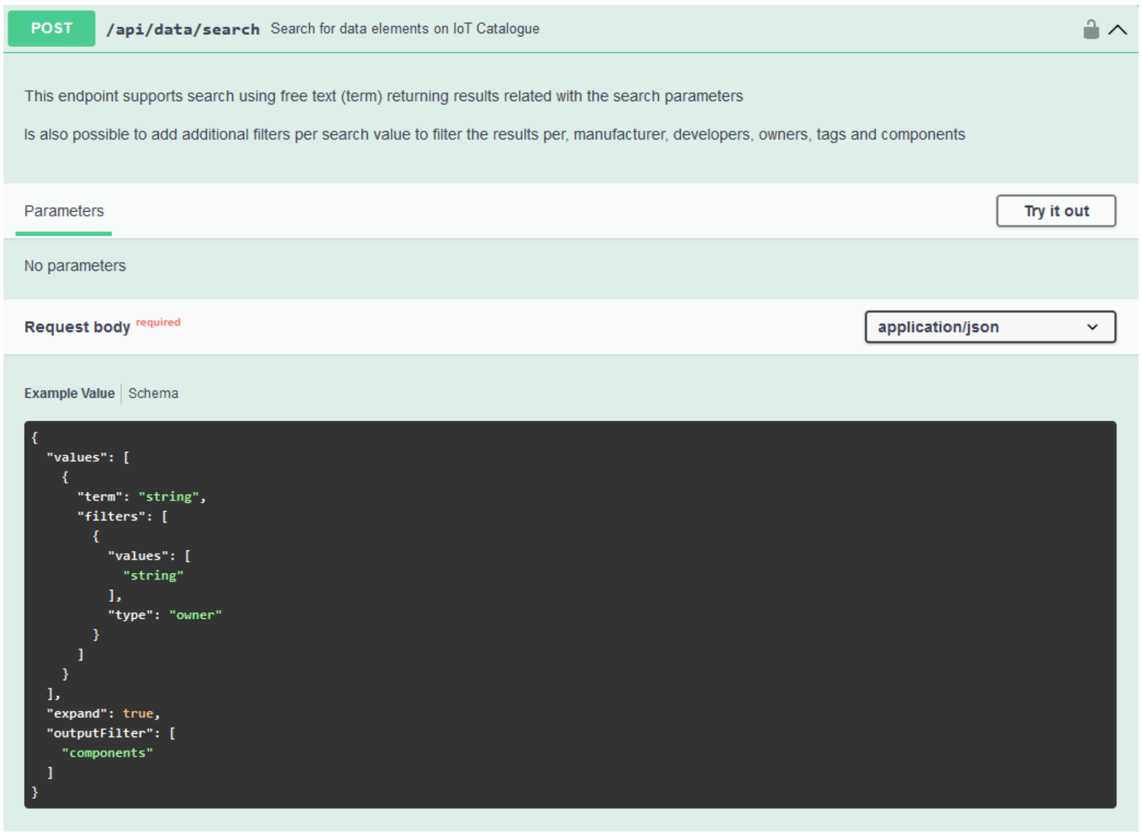
2.3.3. API Endpoint Structure

The semantic search engine’s API endpoint serves as an intermediary that formulates and directs queries to the Federated Data Asset Catalogue (FDAC). When a user submits a query, the endpoint constructs a structured request in JSON format, capturing the nuances of the user's search intent. Each query consists of key-value pairs where "values" is an array of search criteria.

The "**term**" parameter is the primary string the user wants to search for across the FDAC database. It acts as the core of the search, directing the engine to retrieve assets related to this term. Accompanying the term, "**filters**" can refine search results further. These filters are specified as arrays with a "type" attribute indicating the category of the filter, such as "owner" for asset ownership.

An "**expand**" Boolean flag determines the depth of information returned. If set to true, the engine includes the complete JSON objects of content within the results. When false, only the IDs of the assets are returned, making for a more lightweight data transfer.

Lastly, "**outputFilter**" designates the type of content to return from the query. In this case, setting it to "components" instructs the engine to restrict results to data assets specifically, excluding other content types that may exist within the FDAC.



/api/data/search Search for data elements on IoT Catalogue

This endpoint supports search using free text (term) returning results related with the search parameters

Is also possible to add additional filters per search value to filter the results per, manufacturer, developers, owners, tags and components

Parameters

No parameters

Request body required

application/json

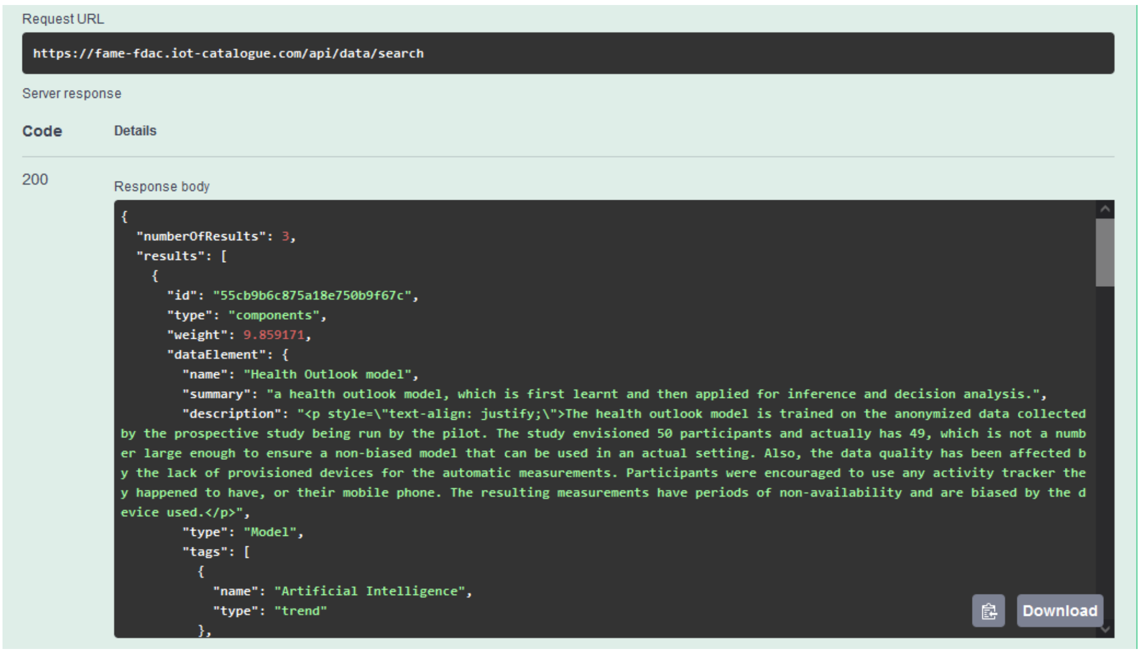
Example Value | Schema

```
{
  "values": [
    {
      "term": "string",
      "filters": [
        {
          "values": [
            "string"
          ],
          "type": "owner"
        }
      ]
    }
  ],
  "expand": true,
  "outputFilter": [
    "components"
  ]
}
```

Figure 23 - Semantic Search Engine API endpoint

2.3.4. Integration with FDAC

The current FDAC integration with the semantic search engine enables it to provide lists of data assets that align with the input search criteria. The engine retrieves and ranks these results based on semantic relevance, using metadata that includes names, summaries, and detailed descriptions. The results are enhanced with tags that categorize and describe each asset's function and application. Developers' details and relevant imagery from the assets' media galleries are also included, alongside visibility status, to present a complete picture of each asset. This structured approach facilitates a user-focused search experience, allowing for both broad and specific queries within the FDAC's diverse range of data assets.



Request URL

```
https://fame-fdac.iot-catalogue.com/api/data/search
```

Server response

Code Details

200 Response body

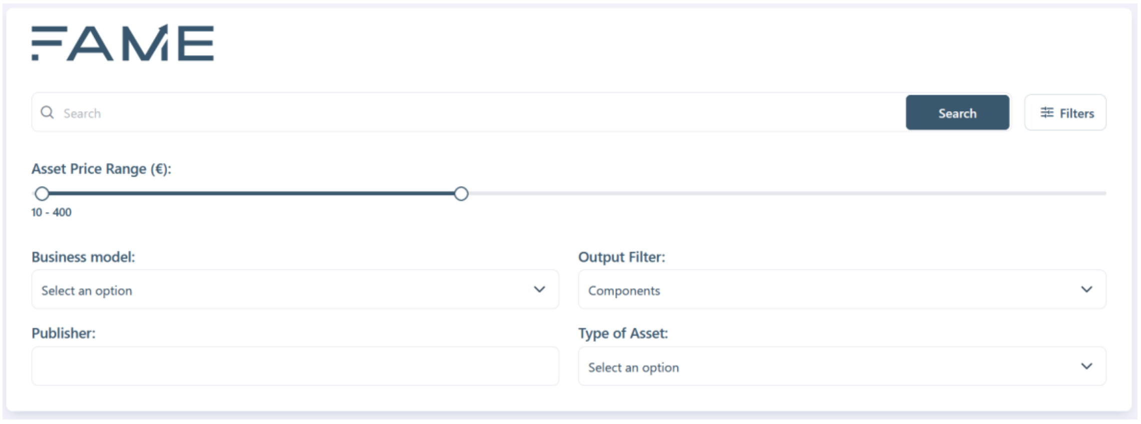
```
{
  "numberOfResults": 3,
  "results": [
    {
      "id": "55cb9b6c875a18e750b9f67c",
      "type": "components",
      "weight": 9.859171,
      "dataElement": {
        "name": "Health Outlook model",
        "summary": "a health outlook model, which is first learnt and then applied for inference and decision analysis.",
        "description": "<p style=\"text-align: justify;\">The health outlook model is trained on the anonymized data collected
by the prospective study being run by the pilot. The study envisioned 50 participants and actually has 49, which is not a numb
er large enough to ensure a non-biased model that can be used in an actual setting. Also, the data quality has been affected b
y the lack of provisioned devices for the automatic measurements. Participants were encouraged to use any activity tracker the
y happened to have, or their mobile phone. The resulting measurements have periods of non-availability and are biased by the d
evice used.</p>",
        "type": "Model",
        "tags": [
          {
            "name": "Artificial Intelligence",
            "type": "trend"
          },
```

Figure 24 - Integrated SSE / FDAC API interface

2.3.5. State of PAT Integration

The integration with the PAT module is in the planning phase. At this stage, the engine is being prepared to handle additional data fields that will be provided by PAT module, like price ranges of assets. Although the exact method of this integration is yet to be finalized, the endpoint will be designed to update seamlessly with pricing information fields to present comprehensive data to the user.

2.3.6. Search Interface & User Journey



Asset Price Range (€):
10 - 400

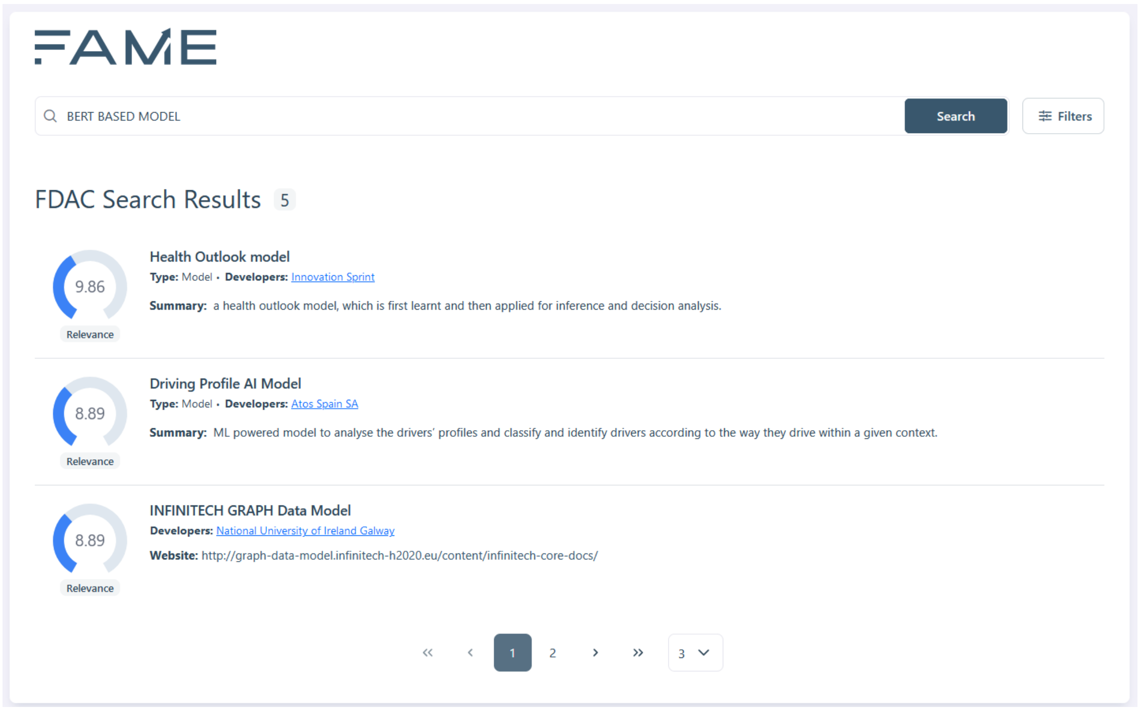
Business model: Select an option

Output Filter: Components

Publisher:

Type of Asset: Select an option

Figure 25 - Semantic Search Interface (1)



FDAC Search Results 5

9.86 Relevance
Health Outlook model
Type: Model • Developers: Innovation Sprint
Summary: a health outlook model, which is first learnt and then applied for inference and decision analysis.

8.89 Relevance
Driving Profile AI Model
Type: Model • Developers: Atos Spain SA
Summary: ML powered model to analyse the drivers' profiles and classify and identify drivers according to the way they drive within a given context.

8.89 Relevance
INFINITECH GRAPH Data Model
Developers: National University of Ireland Galway
Website: http://graph-data-model.infinitech-h2020.eu/content/infinitech-core-docs/

Figure 26 - Semantic Search Interface (2)

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

The semantic search engine's current iteration has been built using Flask, a Python micro-framework that's well-suited for small to medium web applications. The choice of Flask allows for the flexibility and rapid development needed in the early stages of project growth. The frontend interaction is managed through JavaScript, providing a dynamic user interface without overburdening the server side with processing.

The communication between the search engine and the FDAC uses a RESTful API design, with JSON as the data format. This ensures a smooth and effective exchange of data. Python’s Requests library manages the HTTP interactions, and the data processing leverages Python's ecosystem, including libraries like NLTK for natural language processing tasks.

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 is necessary to be inside the root folder with code

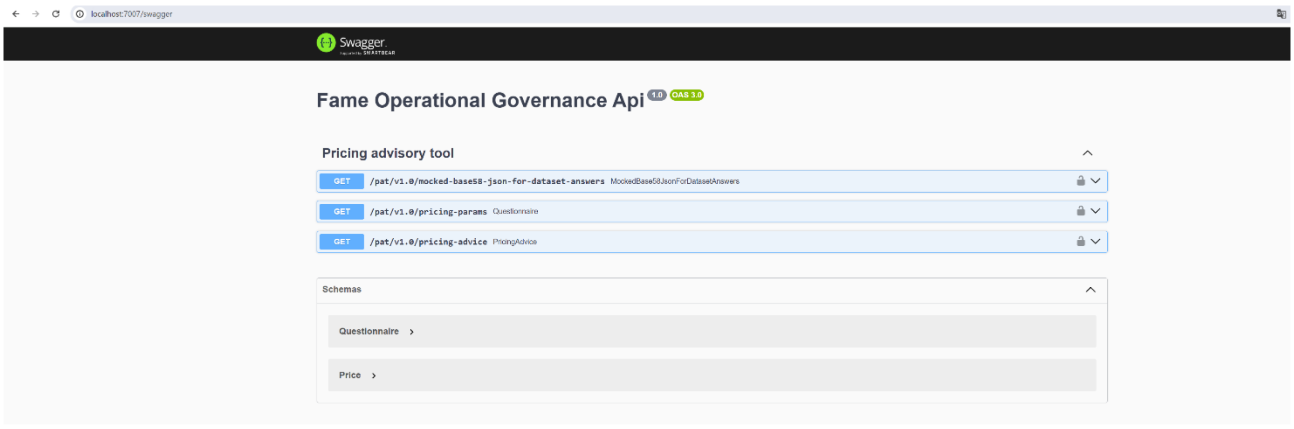
To install all the necessary packages, run the following in terminal:

```
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



Fame Operational Governance Api 1.0 OAS 3.0

Pricing advisory tool

- GET /pat/v1.0/mocked-base58-json-for-dataset-answers MockedBase58JsonForDatasetAnswers
- GET /pat/v1.0/pricing-params Questionnaire
- GET /pat/v1.0/pricing-advice PricingAdvice

Schemas

- Questionnaire
- Price

Figure 27 - Price Advisory API services

3.2 Trading and Monetization 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 is configured to use Docker for creating and managing the application's container. The following npm scripts are provided to simplify Docker operations such as building, pushing, and running the Docker image locally.

Building the Docker Image

To build the Docker image for this project, run:

```
npm run docker:build
```

This command executes a Docker build operation with the tag `harbor.gftinnovation.eu/fame/tm:latest`,, using the Dockerfile located at `devops/Dockerfile`. Ensure you are in the project root directory when running this command.

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 `harbor.infinitech-h2020.eu` registry, run:

```
npm run 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 Docker image locally, use:

```
npm run docker:run
```

This command will start a container named `fame-tm` from the `harbor.infinitech-h2020.eu/fame/tm:latest` image. The application inside the container will be accessible through port 3000 on your local machine, as this port is mapped to port 3000 in the container.

Note: The `--rm` flag is used to automatically remove the container when it is stopped. This helps in avoiding accumulation of stopped containers.

3.2.3. Usage

Deploying Smart Contracts (and upgrading)

To deploy the necessary contracts:

```
npm run deploy:hardhat:besu
```

To upgrade the DataAccess (ERC-1155) logic contract:

```
npm run upgrade:hardhat:besu
```

Running Swagger API (endpoints)

To run:

```
npm run start:dev
```

After that to access the swagger documentation and interactions, use this URL: 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

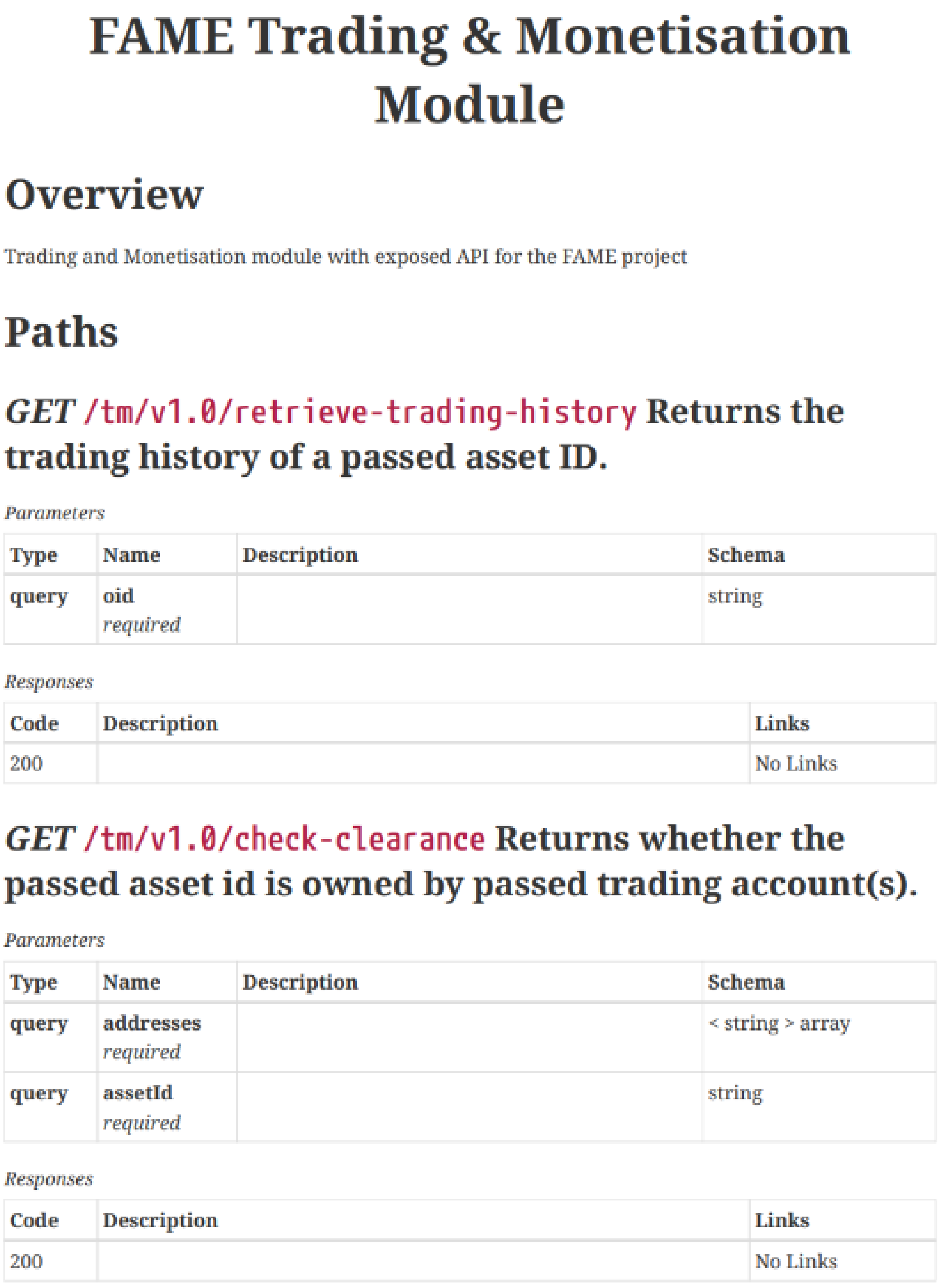
In this document, we described the logic and technical specifications of the first release of 3 modules contributing to the Fame federated marketplace: Trading and Monetisation (T&M), Price Advisory (PAT, SAT) and Semantic Search, all playing an essential role in the Asset Publishing and Offering Definition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

It should be clarified that this is just a "minimum viable product", as it only supports the very basic features of the FAME platform and does not have the technology readiness level that is expected from the final release. The platform will have to gain other key capabilities in the coming months, as all will be combined to the end user in the 'Dashboard' of the FAME integrated marketplace platform. We also need to understand that the overall Fame federated marketplace can only work when combined with a number of administrative features – like digital currency management, catalogue editing, etc. These capabilities will be mostly enabled by other platform modules, in particular P&T and Operational Governance (GOV). The assembly 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") belong.

Annex 1

Components of APIs for Trade & Monetisation

Swagger documentation



FAME Trading & Monetisation Module

Overview

Trading and Monetisation module with exposed API for the FAME project

Paths

GET `/tm/v1.0/retrieve-trading-history` Returns the trading history of a passed asset ID.

Parameters

Type	Name	Description	Schema
query	**oid** *required*		string

Responses

Code	Description	Links
200		No Links

GET `/tm/v1.0/check-clearance` Returns whether the passed asset id is owned by passed trading account(s).

Parameters

Type	Name	Description	Schema
query	**addresses** *required*		< string > array
query	**assetId** *required*		string

Responses

Code	Description	Links
200		No Links

GET `/tm/v1.0/list-cleared-items` Returns the list of access tokens that are owned by the passed trading account.

Parameters

Type	Name	Description	Schema
query	**tradingAccount** *required*		string

Responses

Code	Description	Links
200		No Links

POST `/tm/v1.0/submissions/order` Submit purchasing order [Open API], Returns an unsigned transaction

Responses

Code	Description	Links
201		No Links

POST `/tm/v1.0/offerings` Setup trading (create/mint an offering token)!

Responses

Code	Description	Links
201		No Links

DELETE `/tm/v1.0/remove-offering` Remove (burn) an offering token.

Parameters

Type	Name	Description	Schema
query	**oid** *required*		string

Responses

Code	Description	Links
200		No Links

GET `/gov/v1.0/{tid}` Get trading account payment token balance

Parameters

Type	Name	Description	Schema
path	**tid** *required*		string
query	**token** *required*		string

Responses

Code	Description	Links
200		No Links

POST `/gov/v1.0/{tid}` Post Trading Account, MINT/BURN payment tokens

Parameters

Type	Name	Description	Schema
path	**tid** *required*		string

Responses

Code	Description	Links
201		No Links

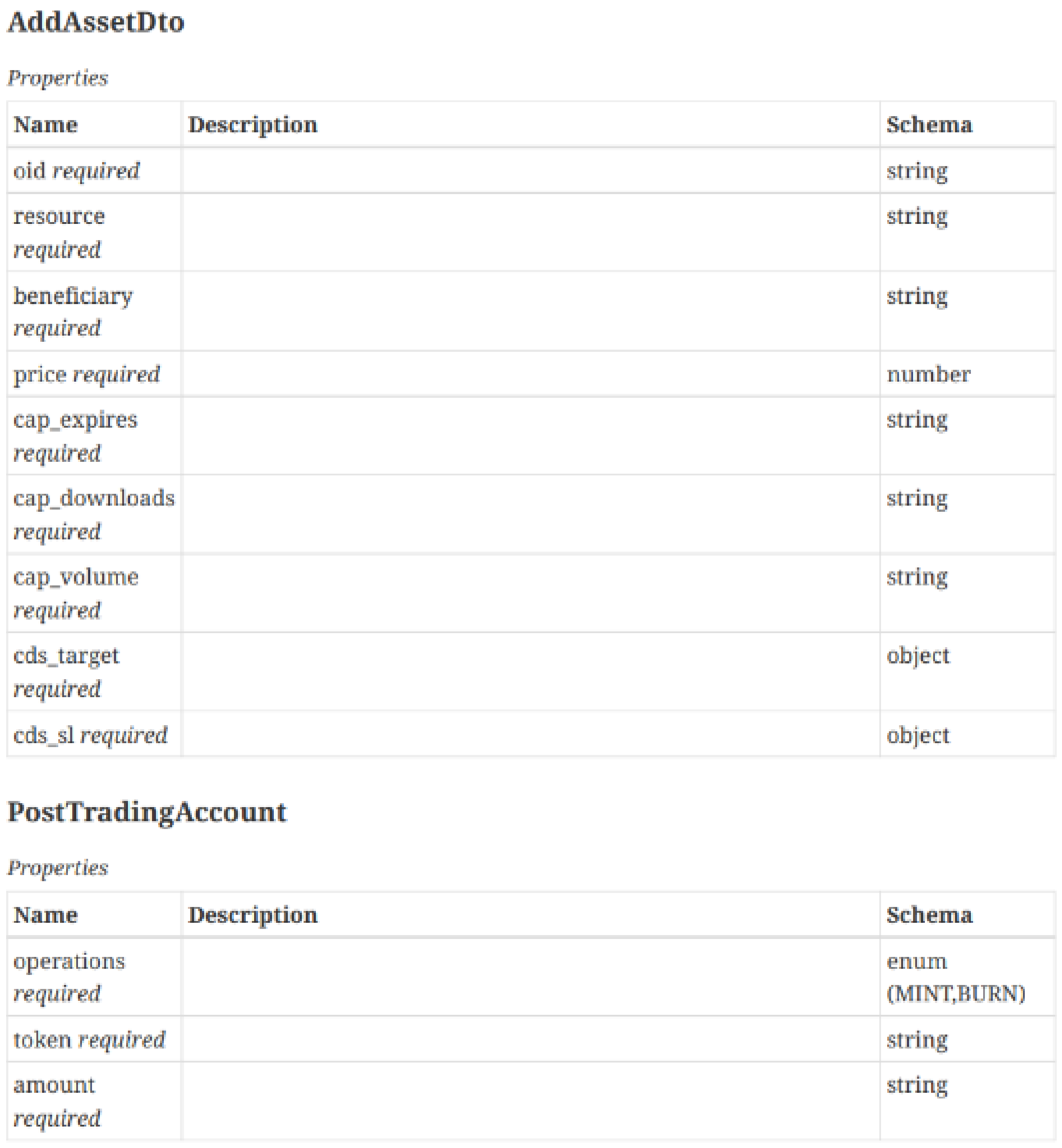
Components

Schemas

PurchaseAccessRightDto

Properties

Name	Description	Schema
oid *required*		string



AddAssetDto

Properties

Name	Description	Schema
oid *required*		string
resource *required*		string
beneficiary *required*		string
price *required*		number
cap_expires *required*		string
cap_downloads *required*		string
cap_volume *required*		string
cds_target *required*		object
cds_sl *required*		object

PostTradingAccount

Properties

Name	Description	Schema
operations *required*		enum (MINT,BURN)
token *required*		string
amount *required*		string