D4.4 - Blockchain-based Data Provenance Infrastructure

Title	D4.4 - Blockchain-based Data Provenance Infrastructure
Revision Number	1.0
Task reference	T4.1
Lead Beneficiary	ENG
Responsible	
Partners	FTS, INNOV, TRB, UBI
Deliverable Type	DEM
Dissemination Level	PU
Due Date	2024-12-31 [Month 24]
Delivered Date	2025-01-20
Internal Reviewers	UBI IDSA
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	2024-12-02	ENG	Table of Contents
0.2	2024-12-19	FTS, INNOV, TRB	Integrated version with contributions
0.3	2025-01-10	UBI IDSA	Version for peer review
0.4	2025-01-16	ENG	Update after peer review
1.0	2025-01-20	ENG	Version for submission

Definitions

Acronyms	Definition
AAI	authentication authorization infrastructure
AID	Asset Identifier
API	Application Programming Interface
APM	Assets Policy Management
AWS	Amazon Web Services
BIC	Business Identifier Code
CDS	Copernicus Data Store
DCAT	Data Catalog Vocabulary
DUN	Data Universal Numbering System
FAME	Federated decentralized trusted dAta Marketplace for Embedded finance
FDAC	Federated Data Assets Catalogue
GOV	Operational Governance
JSON	JavaScript Object Notation
LEI	Legal Entity Identifier
MVP	Minimum Viable Product|Platform
OID	Offering Identifier
REST	Representational State Transfer
RPC	Remote Procedure Call
SA	Supervisory Authority
SHA3	Secure Hash Algorithm v3
SLA	Service Level Agreement
SQL	Structured Query Language
TB	Terabyte (1,000 gigabytes)
TID	Trade Account Identifier
TIN	Taxpayer Identification Number
TR	Technical Requirement
UI	User Interface
UID	User Identifier
UML	Unified Modelling Language|Universal Markup Language
URL	Uniform Resource Locator

Executive Summary

This is the documentation of the software released as deliverable D4.4 “Blockchain-based Data Provenance Infrastructure v2”. It is the second and final instalment of the Provenance and Tracing module (P&T in brief) of the FAME Platform, with new features and improved maturity over the first version that was released as D4.1 at M12. The P&T module provides a public API and some internal integration endpoints. It represents the concrete outcome of task T4.1 “Decentralized Data Provenance and Traceability”, and – together with the Authentication and Authorization Infrastructure provided by task T3.1 – is the main *enabler of trust* for the FAME Marketplace, thanks to the use of Blockchain and smart contract technology: only assets that have been published through the P&T API are considered *trusted assets* in the FAME Federation – and visually identified as such when navigating the catalogue through the FAME Marketplace front-end.

This document includes a very detailed technical specification of each of the P&T module’s components (with a special focus on the API) and an overview on the software implementation and evidence of concrete results. **It is not just an update to the previously released document, but a complete rewriting**.

In its first release, P&T supported only two workflows: **Asset publishing** and Offering definition (the latter now renamed to **Offering publishing** for better consistency). The current version includes some updates to these workflows and adds five new ones: **Asset revision, Asset unpublishing, Offering unpublishing, Trusted Source onboarding, Trusted Source offboarding**. In terms of standalone (i.e., not workflow-related) functionalities, this version adds four new operations to the existing two. Overall, this enhanced feature set meets all the provenance and tracing requirements that were identified for the final Platform.

The software has already reached a sufficient level of maturity to be used outside of the scope of the FAME project, but more improvements will be made during the very last phase, when the cross-module integration will be extensively tested in the scope of task T2.3 “Data Marketplace Platform Integration”. Further evolution of the module, beyond the end of the project, will be made possible thanks to a business-friendly open-source license.

Table of Contents

List of Figures

1 Introduction

1.1 Objective of the Deliverable

This document is a simple *factsheet* describing the software prototype released as deliverable D4.4 "Blockchain-based Data Provenance Infrastructure v2". This is the final release of the series, which improves over the first one – D4.1, defined as a *minimum viable product* at the time – by adding new features and improving code maturity.

The D4.4 prototype has been integrated, tested and demonstrated as part of the FAME Platform. This demonstrator has the capability of publishing assets to the FAME federated marketplace and allows publishers to define commercial offerings for them. **With respect to the previous version, this prototype adds support for catalogue management capabilities, like entry editing and unpublishing, and is better integrated with the rest of the Platform.**

1.2 Insights from other Tasks and Deliverables

Deliverable D4.4 is the result of activities performed in the scope of task T4.1 "Decentralized Data Provenance and Traceability". This task has a very close relationship with other tasks, namely T3.3 Federated Catalogue of Data Assets, T4.3 "Smart Contracts for Programmable Trading and Monetization" and T4.5 "Business and Operational Models for the FAME Marketplace"; it is also related with the work done in T2.2 Platform Architecture and Technical Specifications, T2.3 Data Marketplace Platform Integration, T3.1 Federated AAI Infrastructure, T3.2 Unified Security Policy Management, and T4.2 "Accounting, Trading, Pricing and Monetization Schemes". This complex network of relationships is depicted in Figure 1 below.

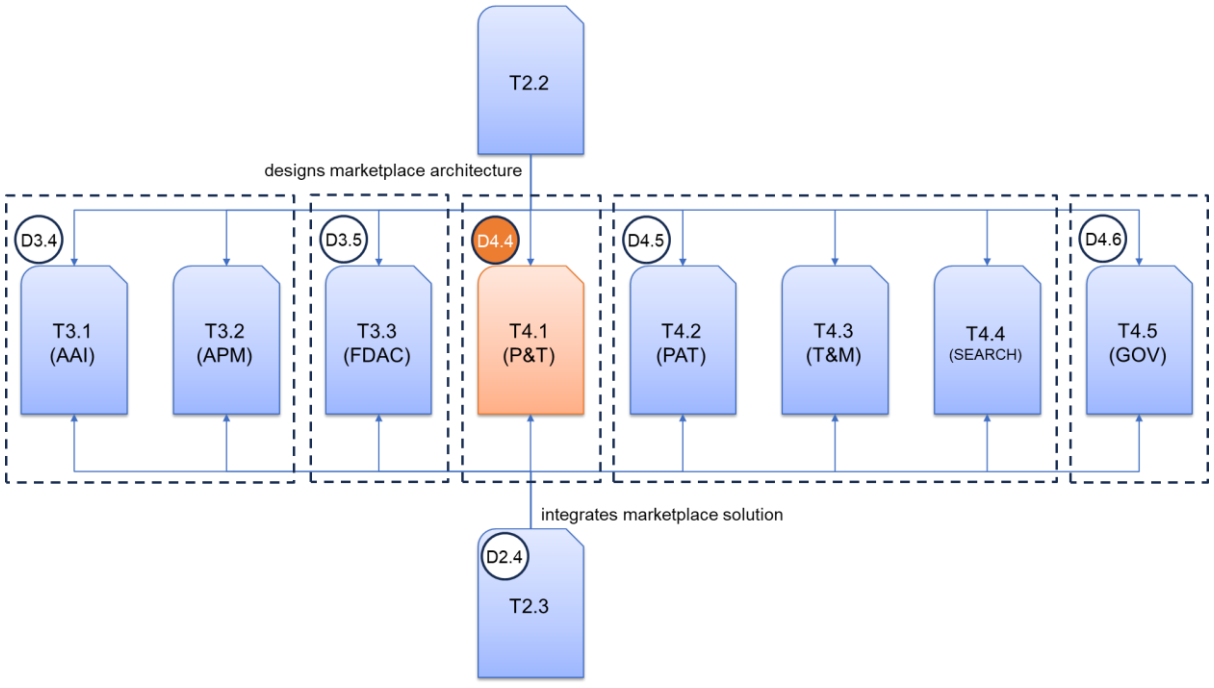

Figure 1 - Task / deliverable relationships (actualized from rel. 1)

The above figure also displays the shorthand notation for the software modules of the FAME Platform. P&T, which stands for Provenance and Tracing, is the topic of this deliverable. **In the rest of the document, "Provenance and Tracing" and "P&T" are used interchangeably to represent the "Blockchain-based Data Provenance Infrastructure"**, as this module was originally called in the Description of Action document. This change to a terser name was suggested by the need to better reflect the role of the module within the general architecture of the FAME Platform.

1.3 Structure

This document consists of five sections.

1. Introduction – This section.
2. Module Overview – 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 – 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 – Gives an overview on the software implementation and evidence of concrete results, by including references to source code and screenshots from the online system.
5. Conclusions – Contains a recap of the achievements and outlook.

It is important to point out that this document is an incremental evolution of the one released with the first version of the P&T prototype (D4.1): all the content that is still relevant was kept, new or changed content was added where appropriate. This version can thus be considered as a standalone document, which makes D4.1 obsolete. Given that 90% of impactful changes is contained in the specifications section, we have added a dedicated sub-section there to summarize them – see §3.1 Changes and improvements over the first release.

2 Module Overview

The Provenance and Tracing (P&T) module is, together with the Authentication and Authorization Infrastructure (AAI), the main *enabler of trust* of the FAME marketplace: it ensures the appropriate level of confidence in assets that are published at the federation level. Although the Federated Data Asset Catalogue (FDAC) supports more than one path for asset publishing (e.g., it allows for external dataspaces to *bulk-import* entire catalogues of assets into the FAME marketplace), only the one that is mediated by P&T ensures a high level of trust in the origin and integrity of such assets, thanks to the use of Blockchain and smart contract technology to store and share some highly dependable *root of trust*[1] data. **For this reason, only assets that have been published through the P&T API are considered *trusted assets* in the FAME federation – and visually identified as such when navigating the catalogue through the FAME Marketplace front-end.** In this document, we may refer to these explicitly as "trusted asset", or implicitly as "asset", but the meaning is always the same: assets that are brought into the FAME federation by means of the P&T API.

In addition to this key role, P&T also supports the FAME Platform in delivering generic functionality to users. In particular, the module enables the publishment of commercial offerings linked to trusted assets and allows FAME administrators to manage some Blockchain-based registries that are at the foundation of FAME's trust ecosystem. The processes supported by P&T are discussed in §2.3 "Workflows and Integration with other Modules".

2.1 Positioning into FAME SA

In the C4 architecture model of the FAME Solution Architecture, the P&T module is classified as a "container" named "Assets Provenance & Tracing", which is included in the "Transactional Operations" System – see Figure 2. Its role is to support the "Federation Manager" System by tracking the provenance of asset entries, by protecting their metadata integrity once published on the Federated Data Asset Catalogue and by managing an additional metadata layer that supports asset trading. To achieve these goals, it interacts with the Federated Data Asset Catalogue (FDAC), the Operational Governance (GOV), and the Trading & Monetization (T&M) modules.

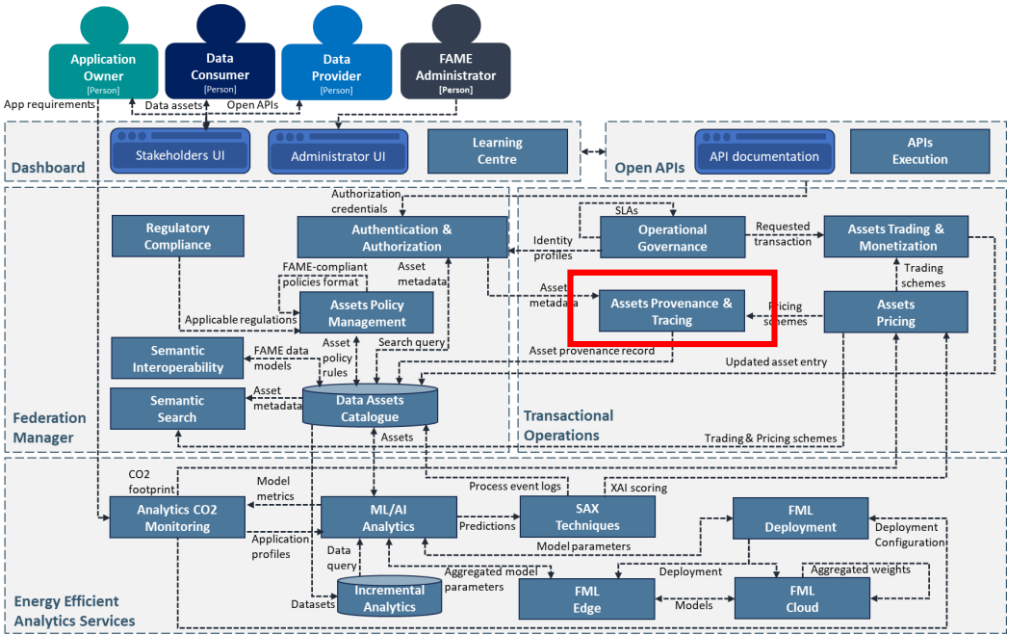

Figure 2 – FAME Solution Architecture: C4 context-level diagram (P&T highlighted)

[1] See https://csrc.nist.gov/glossary/term/roots_of_trust

2.2 C4 Component-level Architecture

In this section we provide the final version of the C4 Component-level diagram for P&T. Previous versions were included in the D2.2 "Technical Specifications and Platform Architecture" deliverable and in the first release of this deliverable. This version – see Figure 3 – reflects the evolution of the FAME Platform's design during of our agile process. With respect to the previously published version, the differences are both cosmetic and substantial.

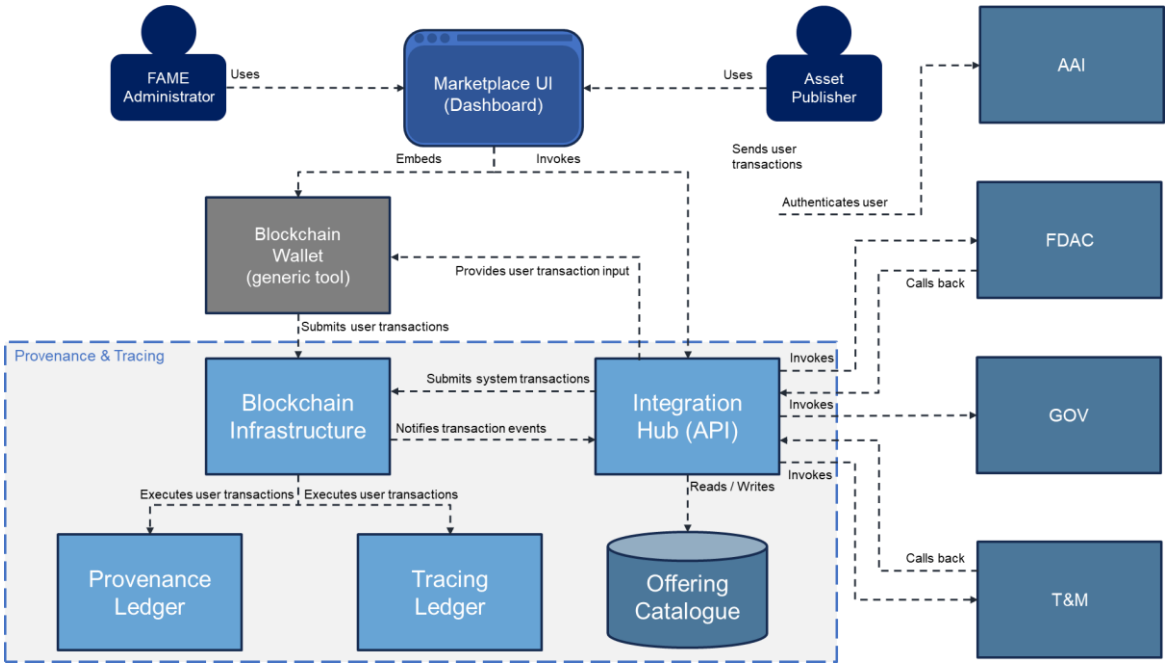

Figure 3 – C4 Component-level diagram (updates since 1st release)

2.3 Workflows and Integration with other Modules

The Integration Hub component is the API layer of the P&T module and where the orchestration of multi-module workflows is implemented. This component provides two distinct API sets: *public* and *internal*. The public API provides service endpoints for user-level operations – i.e., operations that end-users execute directly, or indirectly through the mediation of the FAME Marketplace front-end (also known as the FAME Dashboard). The internal API, on the other hand, is only available to the other modules of the Platform, and are typically used as *callback functions* in multi-module workflows that are rooted in P&T. Another fundamental difference between the two layers is that public API endpoints are executed in the scope of a *security context* – thanks to their integration with the Authentication and Authorization Infrastructure – which carries key information about the calling user: **identity**, **affiliation**, and **role**.

P&T, through the Integration Hub component, supports five cross-module workflows plus two local workflows. These are the following:

1. Asset publishing – To publish a new trusted asset on marketplace.
2. Asset revision – To change the published catalogue entry of a trusted asset. **[NEW]**
3. Asset unpublishing – To remove a trusted asset from the marketplace. **[NEW]**
4. Offering publishing – To associate a commercial offering to a trusted asset.
5. Offering unpublishing – To remove a commercial offering from the marketplace. **[NEW]**
6. Trusted Source onboarding – To qualify a publishing organisation as a trusted source. **[NEW]**
7. Trusted Source offboarding – To revoke the qualification of trusted source. **[NEW]**

Workflows from #1 to #5 are cross-module, meaning that they involve other modules beyond P&T. They can be initiated by any authenticated user of the FAME Platform, provided they meet certain criteria[2]. Below, these five workflows are documented as UML interaction diagrams.

Workflows #6 and #7 are much simpler, being local to P&T. They are only available to users having the administrator role.

In addition to the workflows, there are some standalone functionalities that the Integration Hub provides to its users. Some of them also involve multiple modules. For instance:

A. Identify the commercial offerings linked to a given trusted asset
B. Get the complete definition of a given commercial offering
C. Identify the trusted assets belonging to a given publisher **[NEW]**
D. Get the metadata for a given trusted asset (including an integrity check) **[NEW]**
E. Identify all the trusted sources (only for administrators) **[NEW]**
F. Get the certified contact information of a given trusted source **[NEW]**

It is worth noting that in the first release of the P&T module (D4.1), only workflows #1 and #4 plus functionalities A and B were implemented. Some internal changes have also occurred to these workflows, which are reflected in the updated UML diagrams.

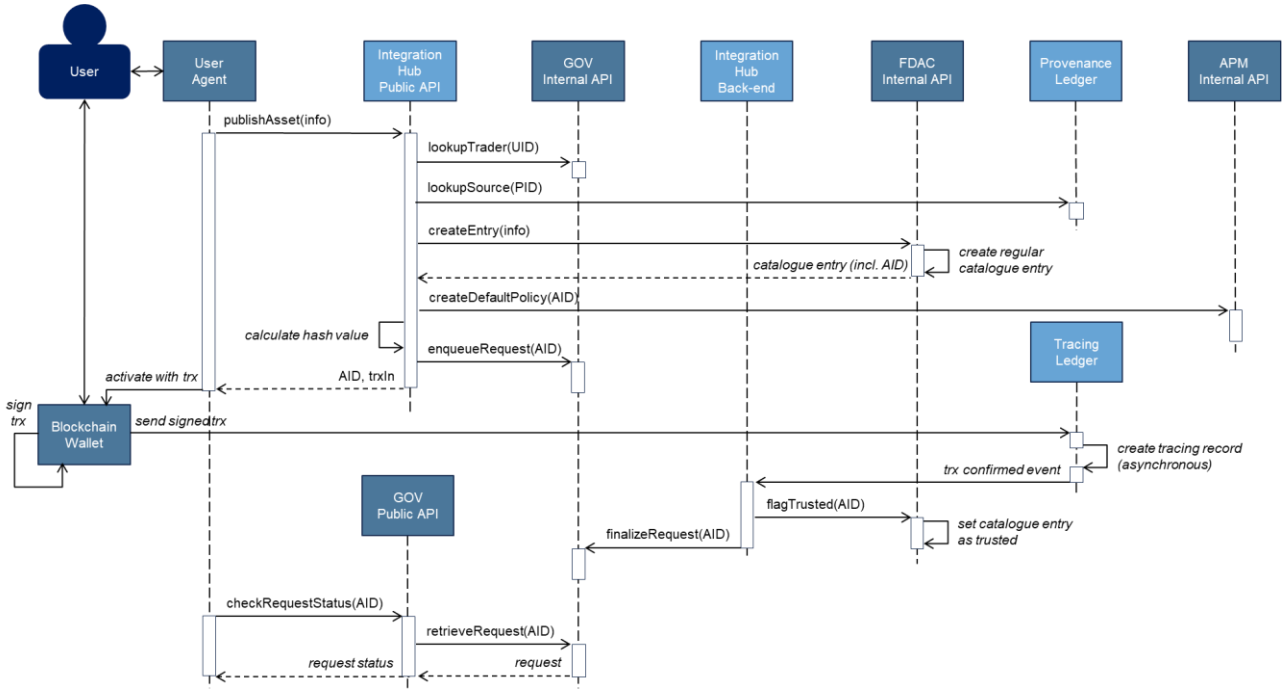

Figure 4 - Asset publishing workflow (updated since 1st release)

[2] Namely, that they are affiliated with a qualified trusted source and that they own an *enrolled* trading account (for more details on the latter point, see the documentation of the Operational Governance module in deliverable D4.3).

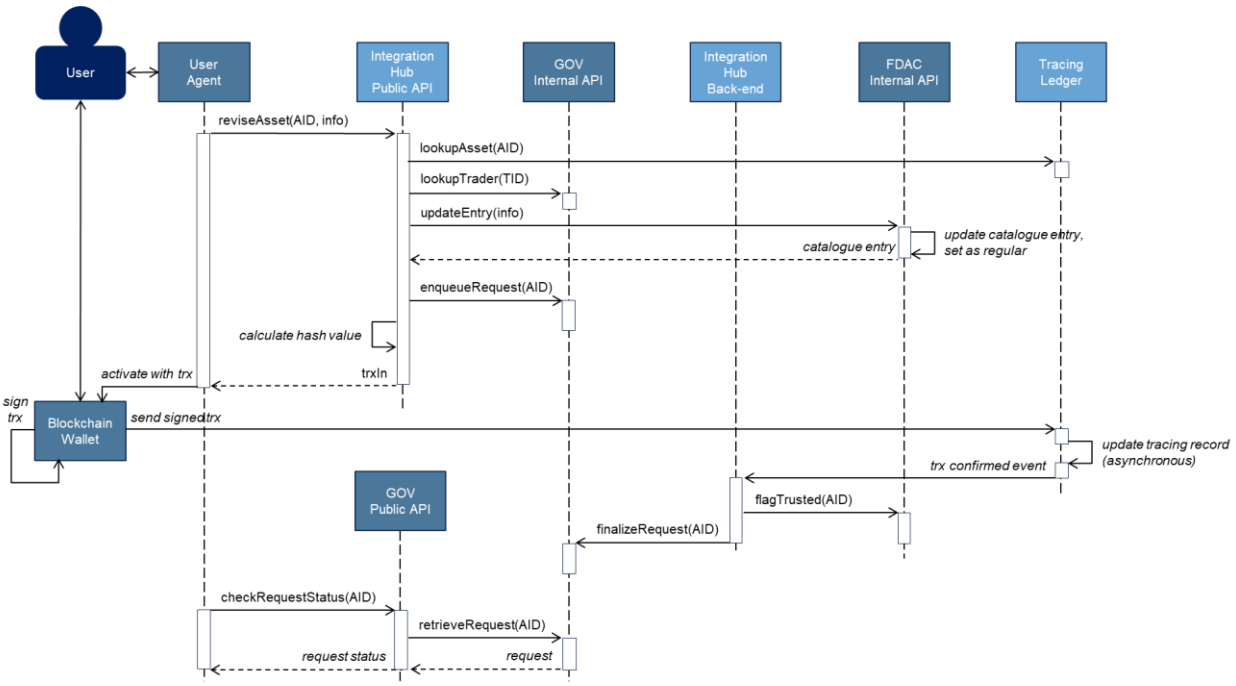

Figure 5 - Asset revision workflow

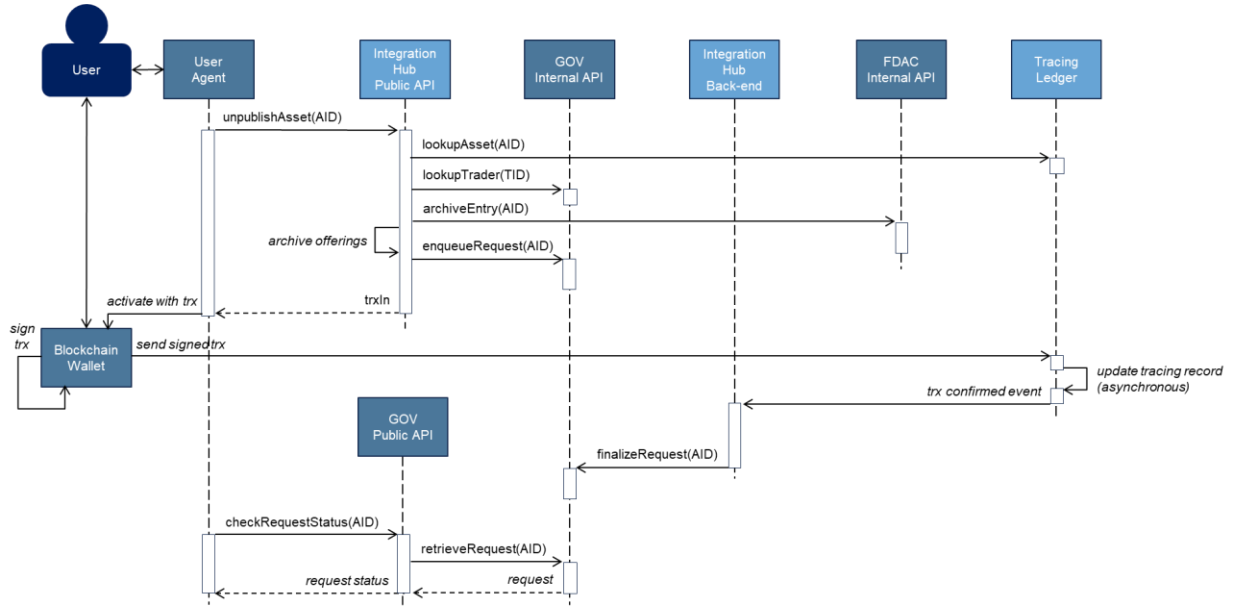

Figure 6 - Asset unpublishing workflow

Asset publishing, revision and unpublishing (Figure 4, Figure 5 and Figure 6, respectively) require that a blockchain transaction is directly signed and submitted to the Tracing Ledger contract by the user who initiated the workflow. The reason for this is that the user/publisher is taking ownership of (and responsibility for) the process, and this fact must be captured into an immutable record on the Tracing Ledger. To meet this nontrivial requirement, the user must have an Ethereum-compatible wallet, must own a Blockchain account and must go through an *enrolment* process that is managed by the Operational Governance module of the FAME Platform. This might seem complex – and it actually is, from an implementation standpoint – but the user experience is quite friendly if the user agent is the FAME Dashboard. The Dashboard takes care of hiding this complexity by integrating the MetaMask wallet[3], so that the user will just have to click on a button to execute the transaction. Once

[3] See https://metamask.io/

submitted, the transaction will be processed in the background, but the Operational Governance module (GOV in the diagram) provides the user with the means for checking the status of their request.

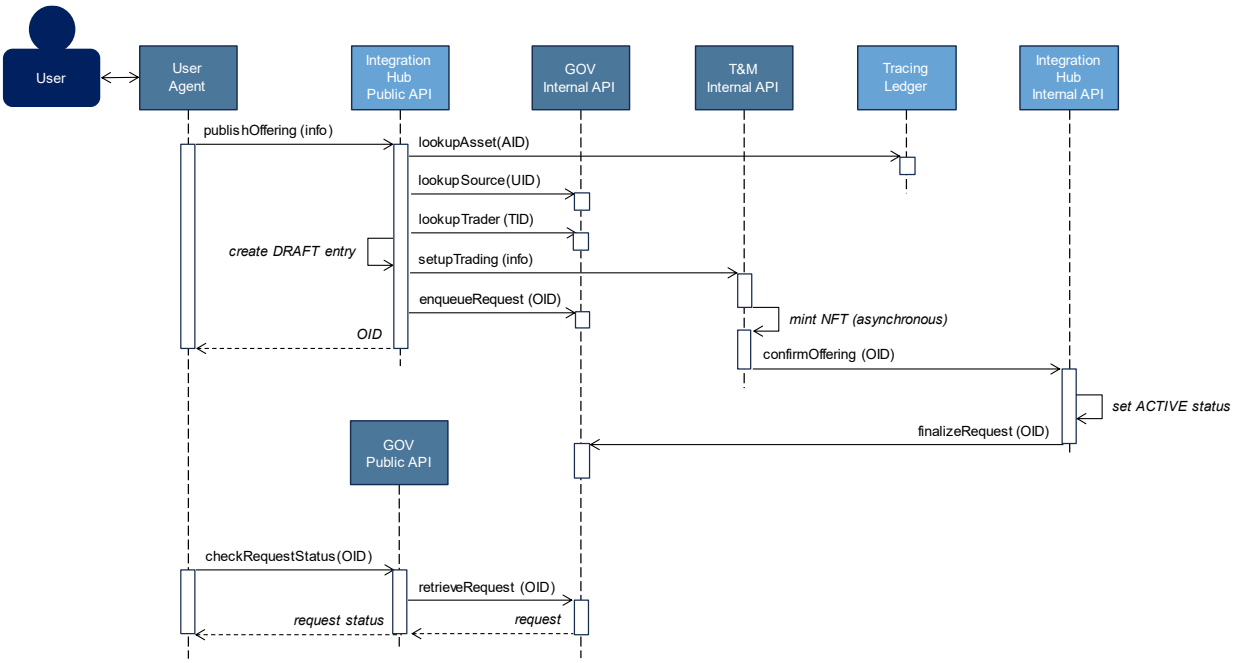

Figure 7 - Offering publishing workflow (updated since 1st release)

Offering publishing (Figure 7) is another case in which a blockchain transaction is triggered. This time, the transaction is signed by a system-owned account, resulting in a simpler user experience that does not rely on any extensions of the user agent. However, as this process is also processed asynchronously with respect to the user's request, a similar mechanism for status checking as seen for asset publishing is in place.

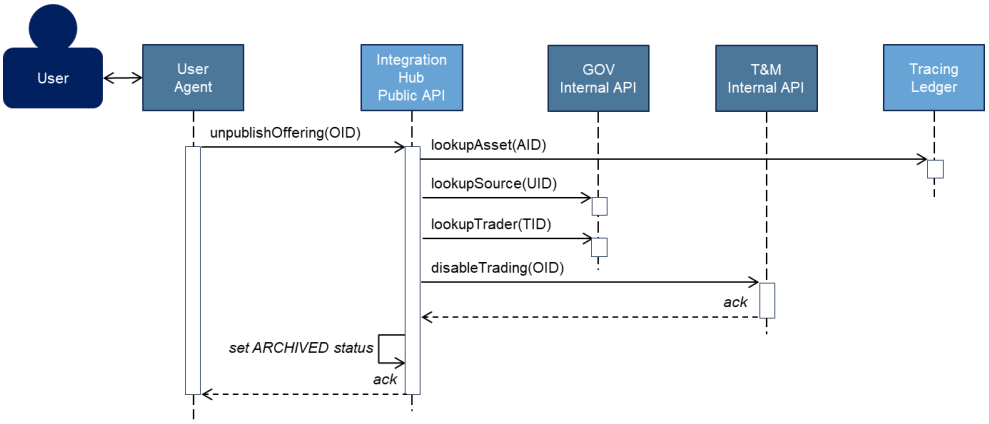

Figure 8 - Offering unpublishing workflow

Offering unpublishing (Figure 8) is the simplest of cases, as the user will get immediate confirmation of the outcome, and no status checking is required – or even supported.

3 Components Specification

In the following sub-sections, the five internal components of the P&T module – i.e., Integration Hub, Blockchain Infrastructure, Provenance Ledger, Tracing Ledger and Offering Catalogue – are analysed individually. The Integration Hub is, by far, the densest chapter, as it describes the fine details of the user-facing API. The other components are discussed very briefly.

3.1 Changes and improvements over the first release

Most changes have occurred in the Integration Hub chapter. Each API operation has now a new format, with a comprehensive functional description, and includes the details of the internal logic. As for the individual operations, none of them was left unchanged since the first release. Operation-wise, these are the changes:

- Submit asset: changed name to Publish asset, updated interface, changes to integration workflow
- **Revise asset: new public operation**
- **List assets for publisher: new public operation**
- **Retrieve asset: new public operation** (replaces Lookup asset with additional functionality)
- Lookup asset: removed as no more necessary
- **Unpublish asset: new public operation**
- Submit offering: changed name to Publish offering, updated interface
- List offerings: updated interface
- Retrieve offering: updated interface
- **Unpublish offering: new public operation**
- Lookup account: removed as no more necessary
- Register source: changed name to Onboard source, visibility changed to public
- **List sources: new public operation**
- Lookup source: changed name to Retrieve source, updated interface, visibility changed to public
- **Offboard source: new public operation**
- Enable account: changed name to Grant permission, updated interface
- **Revoke permission: new internal operation**

Finally, these changes have affected the other chapters:

- Provenance Ledger: added smart contract interface
- Tracing Ledger: added smart contract interface, updated data structure
- Offering Catalogue: updated data structure

3.2 Integration Hub

3.2.1 Description

The Integration Hub is tasked with the coordination of workflows that, while always rooted in P&T, may also involve other modules of the FAME Platform. It provides the API layer of the P&T module, with both public and internal service endpoints (see §2.3), which are documented in §3.2.2.2. **This is the only component of the P&T module exposing an interface: all the others are used internally.**

3.2.2 Technical Specification

3.2.2.1 Baseline Technologies and Tools

The baseline technologies used for the implementation of this component are all Open-Source software. They are listed in the table below.

Table 1 – Integration Hub's baseline technologies

Baseline Technology	Description	Added value to FAME
Node.js	Runtime environment for JavaScript-based network applications	Implementation of the Integration Hub's business logic
NestJS	Framework for exposing Node.js applications as REST-based network services	Implementation of the Integration Hub's API micro-services

IMPORTANT NOTE! Whenever a hash value in mentioned in the specifications, this is always calculated using the SHA3-256 algorithm and then formatted with the Base58Check algorithm.

3.2.2.2 Interfaces

Public API service endpoints are highlighted in **bold**. It is worth noting that these endpoints receive their calls through the FAME Open API layer, which is integrated with the AAI, to the effect that a *security context* with the caller's details (UID, affiliation, roles) is always available to the service's business logic. The same is *not* true for internal API calls, where this information must be provided – if required – by the caller, which is assumed to be *trusted software*.

- **Publish asset** – Initiates the process for publishing a new trusted asset on the marketplace, by creating a matching catalogue entry in the FDAC module and a default policy in the APM module (first phase). The caller must be the owner of an enrolled trading account. This operation returns the input data for a blockchain transaction that targets the Tracing Ledger contract: to complete the publishing process, adding the asset's record to the Tracing Ledger and flagging the catalogue entry as "trusted" (second phase), the caller shall sign and submit the received transaction using an enrolled trading account, which will be permanently linked to the asset as its "owner".
  API Visibility: PUBLIC.
  Access Control: Authenticated user.
  Internal Logic: The service receives the description of an asset that does *not* include the AID (asset identifier) attribute, and checks that the input data is formally correct. The service then checks with the GOV module that the caller owns an enrolled trading account. It then resolves the PID of the user's affiliation in the Provenance Ledger, verifying that the trusted source has "active" status and retrieving the name of the organization. If all checks are passed, the service instructs the FDAC module to create a new entry according to the provided asset description, setting the name of the organization as an additional "creator" attribute. The FDAC assigns to the entry a unique AID and returns to the service the newly created catalogue entry in *normal form*[4]. The service then instructs the APM module to create a default access policy for the asset with the given AID. It then instructs the GOV module to append a new item, identified by the AID and with "pending" status, to the user request queue. The service

[4] The "normal form" of a catalogue entry is a standardized way of formatting the entry's data such that the same data always results in the same computed hash value. This is essential to ensure that the integrity of FDAC catalogue entries can be effectively checked against their matching records in the Tracing Ledger.

then calculates the hash value of the "normalized" catalogue entry. Finally, it prepares the input data (which includes the new AID and the calculated hash value) for a blockchain transaction that targets the Tracing Ledger contract and returns such data to the caller.

Specification:

```
Request: POST https://<public_address>/api/v1.0/assets
Request body (application/json):
{
  (Asset Descriptor object, without the dcterms_identifier field)
}

Response status code: 202 ACCEPTED
Response body (application/json):
{
  "aid": "..",  // asset identifier, as assigned by system
  "hash": "..", // hash value of the catalogue entry, calculated by system
  "addr": "..", // Tracing Ledger contract address
  "abi": ".."   // Tracing Ledger contract interface specs
}
```

Example:

Request body

```
{
  "dcterms_title": "Dummy Asset ABCDEFG",
  "dcterms_description_short": "Lorem ipsum dolor ...",
  "dcterms_description_long": "Lorem ipsum dolor ...",
  "dcterms_type": "dataset",
  "dcat_endpointURL": "https://delivery.fake.com/dummy",
  "dcat_landingPage": "https://info.fake.com/dummy",
  "fame_logo": "https://logos.fake.com/dummy",
  "dcat_keyword": ["keyword1","keword2"],
  "dcterms_conformsTo": ["standard1","standard2"],
}
```

In the example above, all the supported fields have a value. However, the following fields are *optional* and can thus be omitted if not necessary: `dcterms_description_long`, `dcat_landingPage`, `fame_logo`, `dcat_keyword`, `dcterms_conformsTo`. The `dcterms_identifier` field is *not* supported (the AID value is generated by the system and is returned to the caller in the response object - see the example below): if provided in input, it is ignored by the service.

Note that, when this operation returns control to the caller, the asset is published on the catalogue as a "regular" entry (i.e., not flagged as a "trusted"): the Tracing Ledger does not yet contain a matching record, meaning that it would be impossible to do an integrity verification check. The complete the process, the caller must sign and submit the received blockchain transaction data, using their own trading wallet and the credentials of an enrolled trading account. When the service receives from the Tracing Ledger contract the notification that the transaction has been approved, it instructs the FDAC module to flag the asset's catalogue entry as "trusted" and the GOV module to flag the matching user request in the queue as "processed".

- **Revise asset** – Initiates the process for revising the existing marketplace information about a trusted asset, by updating its FDAC catalogue entry with new metadata (first phase). The call must be the owner of the trading account used for the publishing of the target asset. This operation returns the input data for a blockchain transaction that targets the Tracing Ledger

contract: to complete the publishing process, updating the asset's record on the Tracing Ledger and flagging the modified catalogue entry as "trusted" (second phase), the caller shall sign and submit the received transaction using the same trading account used for the publishing of the target asset.

API Visibility: PUBLIC.

Access Control: Authenticated user.

Internal Logic: The service receives the partial description of an already published asset, identified by its AID, containing one or more fields to be updated. The description should *not* include any "type" and "title" information, as these are immutable characteristics of every asset (if present, these fields are ignored). The service first checks that the input data is formally correct and that a matching record with "published" status exists in the Tracing Ledger. It then cross-checks with the Tracing Ledger and the GOV module to verify that the caller owns the trading account that was used for publishing the target asset, and that the account is still enrolled. If all checks are passed, the service instructs the FDAC module to change the matching catalogue entry according to the given asset description. This implicitly clears the "trusted" flag of the entry (the flag will be restored later, once the user executes the blockchain transaction required to update the Tracing Ledger – see the closing note). The FDAC module returns to the service the updated catalogue entry in normal form. Then, the service instructs the GOV module to append a new item, identified by the AID and with "pending" status, to the user request queue. It then calculates the new hash value of the "normalized" catalogue entry. Finally, it prepares the input data (which includes the AID and the new hash value) for a blockchain transaction that targets the Tracing Ledger contract and returns such data to the caller.

Specification:

```
Request: PUT https://<public_address>/api/v1.0/assets/{aid}
Path parameters:
• aid - AID of target asset
Request body (application/json):
{
  (Asset Descriptor object, without the
   dcterms_identifier, dcterms_title and dcterms_type fields)
}

Response status code: 202 ACCEPTED
Response body (application/json):
{
  "aid": "..",  // asset identifier, as assigned by system
  "hash": "..", // hash value of the catalogue entry, calculated by system
  "addr": "..", // Tracing Ledger contract address
  "abi": ".."   // Tracing Ledger contract interface specs
}
```

Example:

Request body

```
{
  "dcterms_description_short": "Lorem ipsum dolor ...",
  "dcterms_description_long": "Lorem ipsum dolor ...",
  "dcat_endpointURL": "https://delivery.fake.com/dummy",
  "dcat_landingPage": "https://info.fake.com/dummy",
  "fame_logo": "https://logos.fake.com/dummy",
  "dcat_keyword": ["keyword1","keword2"],
  "dcterms_conformsTo": ["standard1","standard2"],
}
```

In the example above, all the supported fields have a value. All fields are *optional* (only the provided fields are applied to the catalogue entry), but at least one field must be provided. The `dcterms_identifier`, `dcterms_creator`, `dcterms_title` and `dcterms_type` fields, if present, are ignored as they cannot be modified after the asset is published.

Note that, when this operation returns control to the caller, the catalogue entry has been effectively modified but it is not flagged as "trusted" anymore: the still unmodified Tracing Ledger record is now out of sync with the catalogue entry, meaning that an integrity verification check would fail. To realign the blockchain with the catalogue, the caller must sign and submit the received blockchain transaction data, using their own trading wallet and the credentials of an enrolled trading account. *The trading account used for this operation must be the same trading account used for publishing the asset*[5], otherwise the transaction will not be approved by the Tracing Ledger contract. When the service receives from the Tracing Ledger contract the notification that the transaction has been approved, it instructs the FDAC module to flag the asset's catalogue entry as "trusted" and the GOV module to flag the matching user request in the queue as "processed".

- **List assets for publisher** – Returns a list of references pointing to all the trusted assets published by a given trading account. Note that unpublished assets are not included in the result. Note also that, unless the caller has the administrator role, this operation will always return an empty list for trading accounts that are not owned by the caller.
  API Visibility: PUBLIC.
  Access Control: Authenticated user.
  Internal Logic: The service receives the TID of a trading account. If the user has not the ADM role, the service checks with the GOV module if the given account belongs to the current user. If the check is passed, the service retrieves from the Tracing Ledger the list of records corresponding to trusted assets that have been published by the given TID and still have "published" status. It returns a pageable list of objects, each representing a reference to an asset with a minimal set of attributes (AID, publishing timestamp).

[5] If the user owns multiple accounts, a mismatch is possible if the user selects the wrong account in their wallet, causing the transaction to be rejected. In this case, however, the system is left in a consistent state (outcome of phase one), and the user can repeat the operation from the start – any number of times – to receive a new transaction prompt.

Specification:

```
Request: GET https://<public_address>/api/v1.0/assets
                        ?tid={tid}&l={n}&o={n}
Path parameters:
• tid – TID of publisher trading account
• paging params [OPT]:
   o l – maximum number of results to return
   o o – offset of the first result to return

Response status code: 200 OK
Response body (application/json):
[
  {
    "aid": "..", // AID of trusted asset
    "pub": "..", // timestamp of publishing (ISO8601)
  },
  {
    ..
  }
]
```

- **Retrieve asset** – Retrieves the marketplace catalogue entry of a given trusted asset, together with its provenance identifier (PID), its metadata from the Tracing Ledger, and a "verified" flag that is only set if the catalogue entry has passed the standard integrity check. Note that unpublished assets cannot be retrieved, and that the asset publisher's trading account is included in the result only if the caller has the administrator role.
  API Visibility: PUBLIC.
  Access Control: Authenticated user.
  Internal Logic: The service receives an AID. The service first retrieves the matching record from the Tracing Ledger, verifying that the record exists and has "published" status. It also checks with the APM module if the caller has visibility of the target asset according to the existing policy. It then retrieves the matching catalogue entry, in normal form, from the FDAC module, verifying that the entry exists and has "published" status. If all checks are passed, the service looks up the asset publisher's TID (trading account identifier) in the GOV module, obtaining the PID (provenance identifier) of the asset. It then prepares the response for the caller by composing the various information obtained into a single object, omitting the TID if the caller does not have ADM role. The service calculates the normalized catalogue entry's hash value and checks that it matches the hash value saved in the Tracing Ledger record: if that is the case, it sets the "verified" flag to true. Finally, the service returns the response to the caller. It is worth noting that the catalogue entry, coming from the FDAC, will contain the asset's AID and, possibly, the publisher's name as resolved from the Provenance Ledger during the publishing process (see "Publish asset").

Specification:

```
Request: GET https://<public_address>/api/v1.0/assets/{aid}
Path parameters:
• aid – AID of target asset

Response status code: 200 OK
Response body (application/json):
{
  "entry": {..},         // Asset Descriptor object
  "pid": "..",           // PID of publisher affiliation
  "tid": "..",           // TID of publisher trading account
  "pub": "..",           // timestamp of publishing (ISO8601)
  "mod": "..",           // timestamp of last change (ISO8601)
  "hash": "..",          // hash value of catalogue entry
  "verified": true|false // true if the catalogue entry matches the hash
}
```

- **Unpublish asset** – Initiates the process for removing a trusted asset from the marketplace, by marking both the asset's catalogue entry and any associated offerings as "archived" (first phase). The caller must be the owner of the trading account used for the publishing of the target asset. This operation returns the input data for a blockchain transaction that targets the Tracing Ledger contract: to complete the unpublishing process and flag the trusted asset's Tracing Ledger record as "unpublished" (second phase), the caller shall sign and submit the received transaction using the same trading account used for the publishing of the target asset. Note that this operation, although it does not physically delete any records, cannot be reverted.
  API Visibility: PUBLIC.
  Access Control: Authenticated user.
  Internal Logic: The service receives an AID. The service first checks that a matching record with "published" status exists in the Tracing Ledger. It then cross-checks with the Tracing Ledger and the GOV module to verify that the caller owns the trading account that was used for publishing the target asset, and that the account is still enrolled. If all checks are passed, the service instructs the FDAC module to archive the catalogue entry of the asset and, after receiving confirmation that the operation was successful, assigns the "archived" status to all the offerings in the Offering Catalogue that are linked to the asset. Then, the service instructs the GOV module to append a new item, identified by the AID and with "pending" status, to the user request queue. Finally, it prepares the input data (which includes the AID) for a blockchain transaction that targets the Tracing Ledger contract and returns such data to the caller.
  Specification:

```
Request: DELETE https://<public_address>/api/v1.0/assets/{aid}
Path parameters:
• aid - AID of target asset

Response status code: 202 ACCEPTED
Response body (application/json):
{
  "aid": "..",  // asset identifier, as assigned by system
  "addr": "..", // Tracing Ledger contract address
  "abi": ".."   // Tracing Ledger contract interface specs
}
```

  Note that, when this operation returns control to the caller, all catalogue entries related to the target assets have been removed from the marketplace, but the Tracing Ledger still contains

a (now useless) matching record. To realign the blockchain with the state of the marketplace, the caller must sign and submit the received blockchain transaction data, using their own trading wallet and the credentials of an enrolled trading account (see "Submit asset revision"). When the service receives from the Tracing Ledger contract the notification that the transaction has been approved, it instructs the GOV module to flag the matching user request in the queue as "processed". Note however that this second phase of the process is basically just for proper housekeeping, as it does not have any concrete impact on marketplace operations and/or the integrity of information.

- **Publish offering** – Launches the process for publishing a new offering on the marketplace for an existing trusted asset, by creating a temporary entry in the Offering Catalogue and by communicating the details of the offering to the T&M module (first phase). The caller must be affiliated to the same organization that originally published the asset. The process will then continue in the background and will be completed asynchronously (second phase) without any further interaction by the caller.
  API Visibility: PUBLIC.
  Access Control: Authenticated user.
  Internal Logic: The service receives the description of a commercial offering referred to a published trusted asset. The service first checks that the input data is formally correct, and then cross-checks with the Tracing Ledger and the GOV module to verify that the target asset is published and trusted, that the caller is affiliated with the asset's publisher (the PID retrieved from the call security context matches the source of the asset) and that the beneficiary trading account provided in input is also associated with the asset's publisher. If all checks are passed, the service creates a temporary entry in the Offering Catalogue with "draft" status. Such entry is identified by an OID (offering identifier) consisting of the hash value of the offering description in *normalized form*[6]. It then instructs the GOV module to append a new item, identified by the OID and with "pending" status, to the user request queue. The service then instructs the T&M module to setup the trading environment for the offering. Finally, it returns the generated OID to the caller.
  Specification:

```
Request: POST https://<public_address>/api/v1.0/offerings
Request body (application/json):
{
  (Offering Descriptor object)
}

Response status code: 202 ACCEPTED
Response body (application/json):
{
  "id": ".."  // OID of offering assigned by the system
}
```

  Note that, at this point, the offering is *not* yet active: the T&M module must call the "Confirm offering" service endpoint to confirm that the trading environment has been initialized, thus promoting the offering to "active" (phase two). However, the T&M module may call the "Reject offering" service endpoint instead, to the effect that the submitted offering is discarded. In both cases, the service will instruct the GOV module to flag the matching user request in the queue according to the outcome.

[6] This ensures that, when an OID is used as a reference in a trading transaction, the integrity of the matching Offering Descriptor can be verified regardless of where and how it is stored.

- Confirm offering – Changes the status of an existing entry in the Offering Catalogue from "draft" to "published".
  API Visibility: Internal.
  Access Control: N/A.
  Internal Logic: The service receives an OID. If the OID corresponds to an entry in the Offering Catalogue having "draft" status, the service sets the status of the entry to "published" and instructs the GOV module to set the status of the original user request in the queue (matched by OID) to "processed".
  Specification:
  `Request: PUT http://<private_address>/pt/v1.0/offerings/{oid}/confirm`
  `Path parameters:`
  - `oid – OID of target offering`

  `Response status code: 204 NO CONTENT`

- Reject offering – Deletes an existing entry with "draft" status from the Offering Catalogue.
  API Visibility: Internal.
  Access Control: N/A.
  Internal Logic: The service receives an OID and an error description. If the OID corresponds to an entry in the Offering Catalogue having "draft" status, the service deletes the entry and instructs the GOV module to set the status of the original user request in the queue (matched by OID) to "error", appending the error description to the request.
  Specification:
  `Request: PUT http://<private_address>/pt/v1.0/offerings/{oid}/reject`
  `Path parameters:`
  - `oid – OID of target offering`

  `Response status code: 204 NO CONTENT`

- **List offerings for asset** – Returns a list of references pointing to all the active offerings for a given trusted asset.
  API Visibility: PUBLIC.
  Access Control: Authenticated user.
  Internal Logic: The service receives the AID of the root asset. It checks with the APM module if the caller has visibility of the root asset according to the existing policy. If the check is passed, the service returns a pageable list of objects, each representing an individual record in the Offering Catalogue with a minimal set of attributes (OID, title, price, status). Only records with "published" status are included.

Specification:

```
Request: GET https://<public_address>/api/v1.0/offerings
                        ?aid={aid}&l={n}&o={n}
Path parameters:
• aid – AID of root asset
• paging params [OPT]:
   o l – maximum number of results to return
   o o – offset of the first result to return

Response status code: 200 OK
Response body (application/json):
[
  {
    "oid": "..",   // OID assigned by the system
    "title": "..", // title of the offering
    "price": ..    // price in FDE units (decimal number)
  },
  {
    ..
  }
]
```

- **Retrieve offering** – Returns the catalogue entry of an active offering.
  API Visibility: PUBLIC.
  Access Control: Authenticated user.
  Internal Logic: The service receives an OID. If the OID corresponds to an entry in the Offering Catalogue and the entry has "published" status, the service checks with the APM module if the caller has visibility of the root asset according to the existing policy. If the check is passed, the service returns the matching entry.
  Specification:

```
Request: GET https://<public_address>/api/v1.0/offerings/{oid}
Path parameters:
• oid – OID of target offering

Response status code: 200 OK
Response body (application/json):
{
  (Offering Descriptor object)
}
```

- **Unpublish offering** – Removes a given offering from the marketplace. The caller must either have the administrator role or must be affiliated to the same organization that originally published the asset. Note that although this operation does not physically delete any record, it cannot be reverted.
  API Visibility: PUBLIC.
  Access Control: Authenticated user.
  Internal Logic: The service receives the OID of the target offering and checks that the offering exists in the Offering Catalogue and has "published" status. If the user does not have ADM role, the service then cross-checks with the Tracing Ledger (trading account that owns the asset) and the GOV module (owner account's link to a trusted source) to verify that the caller is affiliated with the asset's publisher (the PID retrieved from the call security context is the same as the asset's trusted source). If all checks are passed, the service informs the T&M

module that the offering is no longer active and, if the T&M module acknowledges the change, changes the status of the offering in the Offering Catalogue to “archived”.

Specification:

```
Request: DELETE https://<public_address>/api/v1.0/offerings/{oid}
Path parameters:
• oid – OID of target offering

Response status code: 204 NO CONTENT
```

- **Onboard source** – Launches the process for onboarding a trusted source, by assigning a unique provenance identifier (PID) to the source and submitting a blockchain transaction that will create a new Provenance Ledger record. The caller must have the administrative role. The process will then continue in the background and will be completed without any further interaction by the caller. Note that the caller will have to check separately the outcome of this operation.

  API Visibility: PUBLIC.

  Access Control: Authenticated user with ADM role.

  Internal Logic: The service receives the description of a legal entity. If the description is formally valid, the service generates a random PID. Then, using a system-owned blockchain account, it submits a transaction to the Provenance Ledger contract that will create a new record with the desired data and “active” status. The service immediately returns the assigned PID to the caller: the rest of the process will continue in the background.

  Specification:

```
Request: POST https://<public_address>/api/v1.0/sources
Request body (application/json):
{
  (Legal Entity Descriptor object)
}

Response status code: 202 ACCEPTED
Response body (application/json):
{
  "id": ".." // PID assigned by the system
}
```

- **List sources** – Returns a list of references pointing to all the trusted source records in the Provenance Ledger. The caller is required to have administrator role.

  API Visibility: PUBLIC.

  Access Control: Authenticated user with ADM role.

  Internal Logic: The service returns a pageable list of objects, each representing an individual record in the Provenance Ledger with a minimal set of attributes (PID, legal name, registration date, status). If the call includes an optional “active” parameter, the list only contains the records matching the given status.

Specification:

```
Request: GET https://<public_address>/api/v1.0/sources
                                ?status={n}&l={n}&o={n}
Path parameters:
• active [OPT] – filter by status
                 0=inactive, 1=active
• paging params [OPT]:
   o l – maximum number of results to return
   o o – offset of the first result to return

Response status code: 200 OK
Response body (application/json):
[
  {
    "pid": "..",          // PID of trusted source
    "name": "..",         // name of legal entity
    "in": "..",           // timestamp of onboarding (ISO8601)
    "active": true|false // status
  },
  {
    ..
  }
]
```

- **Retrieve source** – Returns the full details of a given trusted source, including its metadata and status, from the Provenance Ledger.
  API Visibility: PUBLIC.
  Access Control: Authenticated user.
  Internal Logic: The service receives a PID. If the PID matches a record in the Provenance Ledger, the service returns a Legal Entity Descriptor object that represents the trusted source.
  Specification:

```
Request: GET https://<public_address>/api/v1.0/sources/{pid}
Path parameters:
• pid – PID of target source

Response status code: 200 OK
Response body (application/json):
{
  "pid": "..",          // PID of trusted source
  "source": {..},       // Legal Entity Descriptor object
  "in": "..",           // timestamp of onboarding (ISO8601)
  "out": "..",          // timestamp of offboarding (ISO8601)
  "active": true|false // status
}
```

- **Offboard source** – Launches the process for offboarding a trusted source, by submitting a blockchain transaction that will set the status of the target record in the Provenance Ledger to "inactive". The caller must have the administrative role. Once successfully launched, the process will continue in the background and will be completed without any further interaction by the caller. Note that the caller will have to check separately the outcome of this operation, and that this change of status cannot be reverted.
  API Visibility: PUBLIC.
  Access Control: Authenticated user with ADM role.

Internal Logic: The service receives the PID of the target source and checks that the source exists and has “active” status. Then, using a system-owned blockchain account, submits a transaction to the Provenance Ledger contract requesting a change of status for target source. It then immediately returns control to the caller: the rest of the process will continue in the background.
Specification:

```
Request: DELETE https://<public_address>/api/v1.0/sources/{pid}
Path parameters:
• pid – PID of target source

Response status code: 202 ACCEPTED
```

- Grant permission – Launches the process for granting blockchain permission to a trading account, by submitting a blockchain transaction that will add the given account to the on-chain list of "permissioned" accounts, if not already present. Once successfully launched, the process will continue in the background and will be completed without any further interaction by the caller. Note that if the account is already in the list, the transaction will still succeed but will have no effects. Note also that the caller will have to check separately the outcome of this operation.
  API Visibility: Internal.
  Access Control: N/A.
  Internal Logic: The service receives a TID. Using a system-owned blockchain account, the service submits a transaction to the *allowlist* contract that will add the TID to the list (or ignore it if already present). The service returns control to the caller immediately: the rest of the process will continue in the background.
  Specification:

```
Request: POST http://<private_address>/pt/v1.0/permissions
Request body (application/json):
{
  "id": ".." // TID of target trading account
}

Response status code: 202 ACCEPTED
```

- Revoke permission – Launches the process for revoking blockchain permission to a trading account, by submitting a blockchain transaction that will remove the given account from the on-chain list of "permissioned" accounts, if present. Once successfully launched, the process will continue in the background and will be completed without any further interaction by the caller. Note that if the account is not in the list, the transaction will still succeed but will have no effects. Note also that the caller will have to check separately the outcome of this operation.
  API Visibility: Internal.
  Access Control: N/A.
  Internal Logic: The service receives a TID. Using a system-owned blockchain account, the service submits a transaction to the *allowlist* contract that will remove the TID from the list (or do nothing if not in the list). The service returns control to the caller immediately: the rest of the process will continue in the background.

Specification:

```
Request: POST http://<private_address>/pt/v1.0/permissions
Request body (application/json):
{
  "id": ".." // TID of target trading account
}

Response status code: 202 ACCEPTED
```

3.2.2.3 Data Structures

- **Asset Descriptor**

JSON literal that describes a digital asset in terms that are backward-compatible with the DCAT standard (see https://www.w3.org/TR/vocab-dcat-3/) but also compatible with how the FDAC module stores its catalogue entries. When an attribute of this object can be mapped to a DCAT attribute having the same (or compatible) semantics, the attribute name has a `dcat_` or `dcterms_` prefix; otherwise, it is prefixed by the `fame_` string.

```
{
  "dcterms_identifier": "..",       /* unique identifier assigned by system (AID) */
  "dcterms_creator": "..",          /* name of publisher, resolved by system */
  "dcterms_title": "..",            /* name of asset */
  "dcterms_description_short": "..",/* short description of asset */
  "dcterms_description_long": "..", /* long description of asset */
  "dcterms_type": "..",             /* content type
    domain: DATASET | DATASTREAM | MODEL | DOCUMENT | MEDIA | SOFTWARE | SERVICE */
  "dcat_endpointURL": "..",         /* content server (URL) */
  "dcat_landingPage": "..",         /* descriptive/support web page (comma-separated list of URLs) */
  "fame_logo": "..",                /* logo of asset (URL) */
  "dcat_keyword": "..",             /* keywords attached to asset */
  "dcterms_conformsTo": ".."        /* relevant standards (comma-separated list of URLs) */
}
```

Only the following elements are mandatory when using this structure as input for publishing a trusted asset: `dcterms_title`, `dcterms_description_short`, `dcterms_type`, `dcat_endpointURL`. The fields `dcterms_identifier` and `dcterms_creator`, if present, are ignored.

- **Offering Descriptor**

JSON literal that describes a commercial offering for a published asset.

```
{
  "oid": "..",     /* unique identifier assigned by system (OID) */
  "asset": "..",   /* link to root asset (AID) */
  "title": "..",   /* commercial name of the offering: human readable short text */
  "price": "..",   /* amount of FAME currency requested by the seller: decimal number */
  "summary": "..", /* summary of scope, cap, licensing, and SLA: human readable text */
  "scope": "..",   /* formal description of content subset: human readable text */
  "cap": {         /* capping of content access */
    "subscription": "..",  /* duration of subscription: {n}hour|day|week|month|year */
    "downloads": ..,       /* maximum number of complete downloads: {n} */
    "volume": ".."         /* maximum number of downloaded bytes: {n}KB|MB|GB|TB */
  },
  "license": "..", /* formal description of terms and conditions: human readable text */
  "sla": "..",     /* formal description of service level: human readable text */
  "tid": "..",     /* link to beneficiary account (TID) */
  "objref": "..",  /* target object reference: instructions to CDS, machine readable text */
  "slaref": ".."   /* SLA reference: instructions to CDS, machine readable text */
}
```

Only the following elements are mandatory when using this structure as input for publishing an offering: `asset`, `title`, `price`, `summary`, `cap`, `license`, `tid`, `objref`. The elements contained in the

`cap` section are mutually exclusive: when the `cap` section is present, it can only contain one element. If the `oid` field is present, it is ignored.

- **Legal Entity Descriptor**

JSON literal that describes a legal entity. It is used in FAME to identify an organization that is registered as a content provider.

```
{
  "LEI": "..",            /* Legal Entity Identifier (alphanumeric, 20) */
  "DUN": "..",            /* Data Universal Numbering System (numeric, 8) */
  "BIC": "..",            /* Business Identifier Code (alphanumeric, 8 or 11) */
  "TIN": "..",            /* Taxpayer Identification Number (country-specific) */
  "LegalName": "..",
  "LegalAddress": {
    "AddressLine1": "..",
    "AddressLine2": "..",
    "AddressLine3": "..",
    "City": "..",
    "Region": "..",
    "Postcode": "..",
    "Country": ".."       /* ISO3166 alpha-2 code */
  }
}
```

Only the following elements are mandatory when using this structure as input for registering a trusted source: `LegalName`, `LegalAddress/AddressLine1`, `LegalAddress/City`, `LegalAddress/Country`. At least one field between `LEI`, `DUN`, `BIC` and `TIN` must be present.

3.3 Blockchain Infrastructure

3.3.1 Description

The Blockchain Infrastructure is a dedicated blockchain network for the FAME project, where smart contracts belonging the P&T, T&M and AAI modules are hosted. It is compatible with the Ethereum standard, meaning that any smart contract developed in FAME may theoretically be deployed on any Ethereum network. However, the dedicated network is of the *permissioned* type: anyone can read the contents of the distributed ledger, but only authorized accounts can submit transactions (i.e., can write to the ledger or deploy smart contracts). For this purpose, an *allowlist* is maintained by the network administrators.

3.3.2 Technical Specification

3.3.2.1 Baseline Technologies and Tools

The baseline technologies used for the implementation of this component are all open-source software. They are listed in the table below.

Table 2 – Blockchain Infrastructure's baseline technologies

Baseline Technology	Description	Added value to FAME
Hyperledger Besu https://www.hyperledger.org/projects/besu	Ethereum-compatible Blockchain platform	Implementation of a *permissioned* Blockchain network with smart contract hosting capabilities
Quorum Explorer https://github.com/Consensys/quorum-explorer	Simple user tool for browsing the contents of an Ethereum-style distributed ledger	Used for testing and demonstration purposes

3.3.2.2 Deployment

Presently, only one RPC service endpoint is available to users. This is currently deployed at this network address: http://chain.fame-horizon.eu:8545

There is also a Quorum Explorer instance that can be reached at this address:

http://chain.fame-horizon.eu:25000/explorer/nodes.

Note however that user credentials are required for access Quorum Explorer, and these are only released to administrators.

3.4 Provenance Ledger

3.4.1 Description

The Provenance Ledger keeps record of the unique identifier and full contact details of *legal entities* that have been registered, by a FAME administrator, as trusted sources. This registry, implemented as a blockchain smart contract deployed on the Blockchain Infrastructure (see §3.3), enables the assignment of an unambiguous and trustworthy *provenance identifier* (PID) to all digital assets that are published on the FAME federation.

3.4.2 Technical Specification

3.4.2.1 Baseline Technologies and Tools

The baseline technologies used for the implementation of this component are all open-source software. They are listed in the table below.

Table 3 – Provenance Ledger's baseline technologies

Baseline Technology	Description	Added value to FAME
Hyperledger Besu's Ethereum Virtual Machine https://besu.hyperledger.org/	Runtime environment for Solidity-based smart contracts	Implementation of the Provenance Ledger's persistence layer
Solidity language https://soliditylang.org/	Programming language for smart contract	Development of the Provenance Ledger contract

3.4.2.2 Interfaces

The Provenance Ledger smart contract exposes three simple functions:

- `registerSource`, to create a new trusted source entry on the ledger
- `lookupSource`, to read an existing trusted source entry from the ledger
- `deactivateSource`, to mark a registered trusted source as "inactive"

The signature of these functions is reported below without comments. As these interfaces are only exposed internally to the FAME Platforms, and the functions can only be executed by a system-owned blockchain account, we do not provide any more detailed documentation here.

```
function registerSource(
      string memory pid,
      string memory contactinfo
) external returns (string memory) { .. }

function lookupSource (
      string memory pid
) external view returns (Source memory) { .. }

function deactivateSource (
      string memory pid
) { .. }

struct Source {
      bytes pid;
      string entity;
      uint registered;
      uint deactivated;
}
```

3.5 Tracing Ledger

3.5.1 Description

The Tracing Ledger is a smart contract, deployed on the Blockchain Infrastructure (see §3.3), that tracks the publishing of content, creating an immutable record on the blockchain for every digital asset that is published on the FAME federation. The record contains a verified link to the content publisher's entry in the Provenance Ledger (see §3.4). Moreover, the record enables the cross-checking of the corresponding catalogue entry in the FDAC module, as it contains the hash value of the asset's description: any alteration of the catalogue can thus be easily detected. Basically, the Tracing Ledger adds *trustworthiness* to the information published on FDAC.

3.5.2 Technical Specification

3.5.2.1 Baseline Technologies and Tools

The baseline technologies used for the implementation of this component are all open-source software. They are listed in the table below.

Table 4 – Tracing Ledger's baseline technologies

Baseline Technology	Description	Added value to FAME
Hyperledger Besu's Ethereum Virtual Machine https://besu.hyperledger.org/	Runtime environment for Solidity-based smart contracts	Implementation of the Tracing Ledger's persistence layer
Solidity language https://soliditylang.org/	Programming language for smart contract	Development of the Tracing Ledger contract

3.5.2.2 Interfaces

The Tracing Ledger smart contract exposes four simple functions:

- `postAsset`, to create a new trusted asset entry on the ledger
- `lookupAsset`, to read an existing trusted source entry from the ledger
- `updateAsset`, to change the hash value of a published trusted asset
- `archiveAsset`, to mark a registered trusted asset as "archived"

The signature of these functions is reported below without comments. As these interfaces are only exposed internally to the FAME Platforms, and the functions can only be executed by a system-owned blockchain account, we do not provide any more detailed documentation here.

```
function postAsset(
      string memory aid,
      string memory tid,
      string memory hash) public { .. }

function lookupAsset(string memory aid) external view
      returns (Asset memory) { .. }

function updateAsset(
      string memory aid,
      string memory hash) public { .. }

function archiveAsset(string memory aid) public { .. }

struct Asset {
      bytes aid;
      string tid;
      string hash;
      uint timestamp;
}
```

3.6 Offering Catalogue

3.6.1 Description

The Offering Catalogue complements the FDAC module, which does not support trading. The Offering Catalogue contains the formal definition of commercial offerings (i.e., terms and conditions, including pricing) on digital content that is published in the FAME federation. The offering identifier (OID) that is assigned here becomes the reference for all trading operations performed by the T&M module.

3.6.2 Technical Specification

3.6.2.1 Baseline Technologies and Tools

The baseline technologies used for the implementation of this component are all open-source software. They are listed in the table below.

Table 5 – Offering Catalogue's baseline technologies

Baseline Technology	Description	Added value to FAME
MongoDB https://www.mongodb.com/	No-SQL database	Implementation of the Offering Catalogue's persistence layer

3.6.2.2 Data Structures

The Offering Catalogue is implemented on a No-SQL database: the offering records are structured documents.

- **Offering Record**

```
{
  "oid": "..",       /* unique identifier assigned by system (OID) */
  "asset": "..",     /* link to root Asset (AID) */
  "offering": {..}, /* Offering Descriptor object */
  "created": "..",  /* timestamp of creation (ISO8601) */
  "archived": "..", /* timestamp of archiving (ISO8601) */
  "status": ".."     /* (0=DRAFT | 1=PUBLISHED | 2=ARCHIVED) */
}
```

4 Module Demonstration

In this second release of the Provenance and Tracing module prototype, the implementation is 100% aligned with the specs. The code base has undergone a major refactoring, which significantly improved the quality of the software from the perspective of robustness, security and documentation. The latter was enhanced by annotating the code with *NestJS/Swagger decorators*, so that it is now possible to navigate online the API documentation having access to the full details of the interfaces, as exemplified by the five screenshots below - Figure 9, Figure 10, Figure 11, Figure 12, and Figure 13. Note however that Figure 13 refers to the "internal" API of the P&T module, which is not accessible from the public Internet in the FAME Platform's deployment environment.

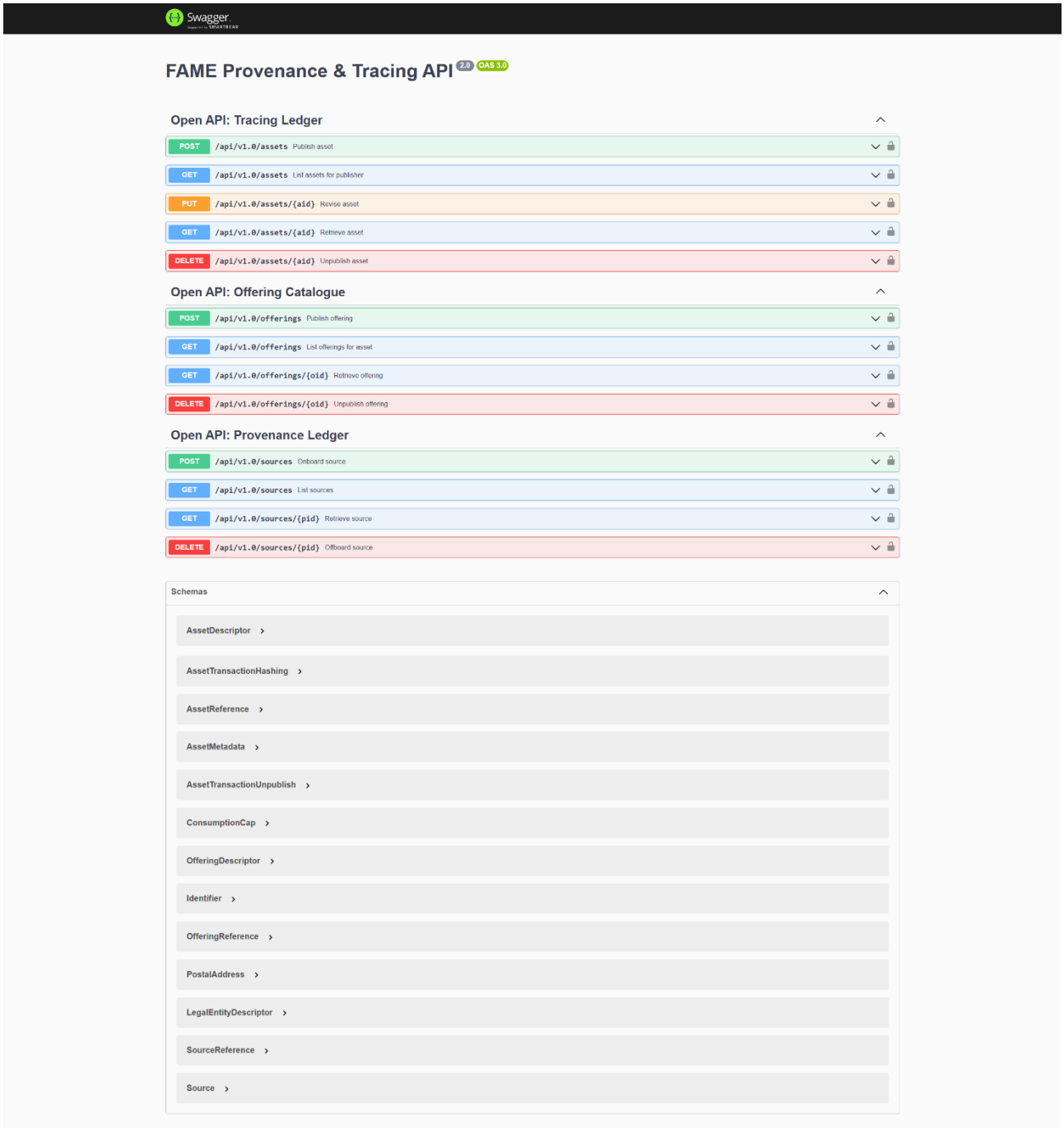

Figure 9 - P&T Open API online documentation, overview (Swagger UI)

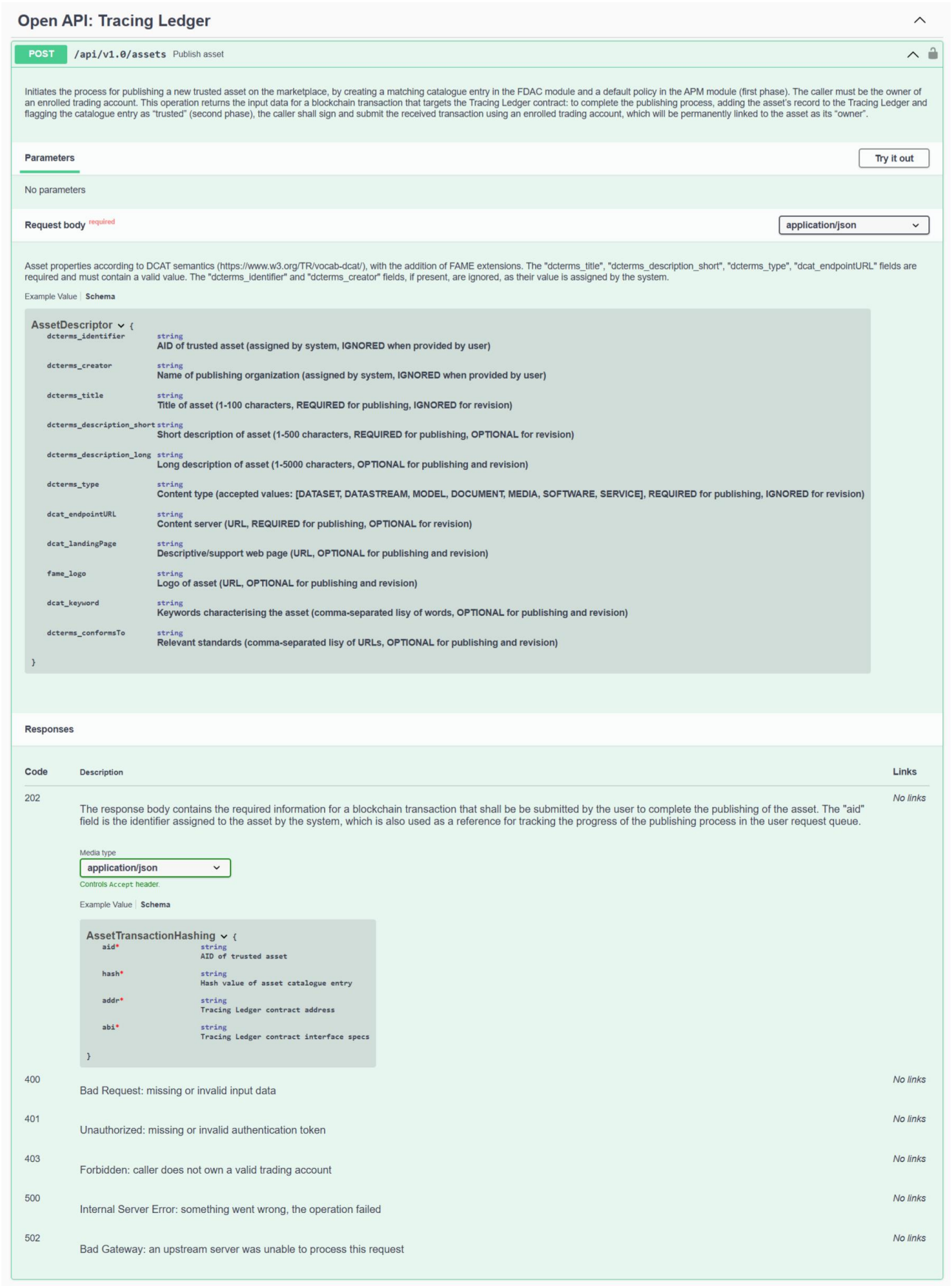

Figure 10 - Publish Asset API online documentation, details (Swagger UI)

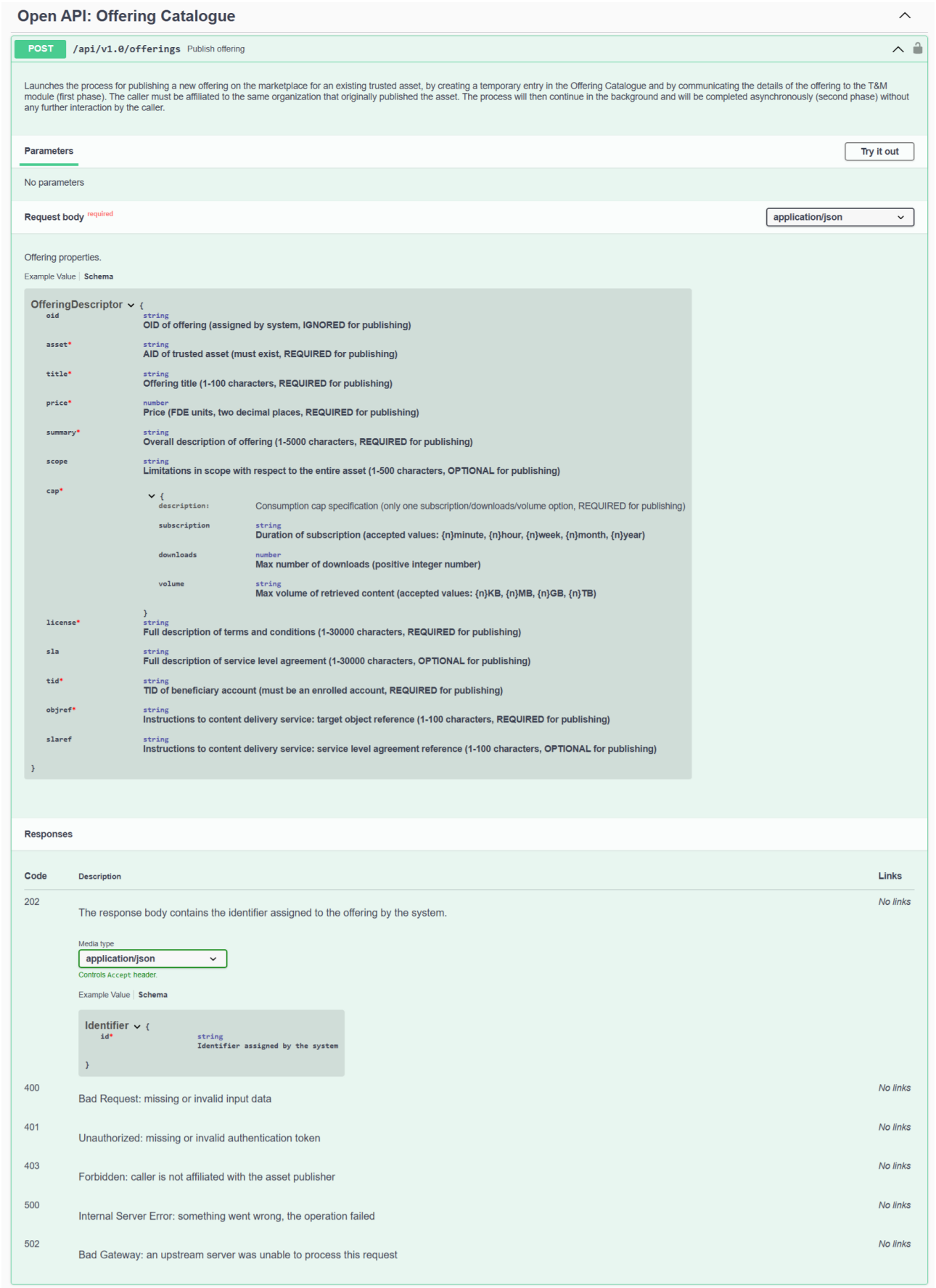

Figure 11 - Publish Offering API online documentation, details (Swagger UI)

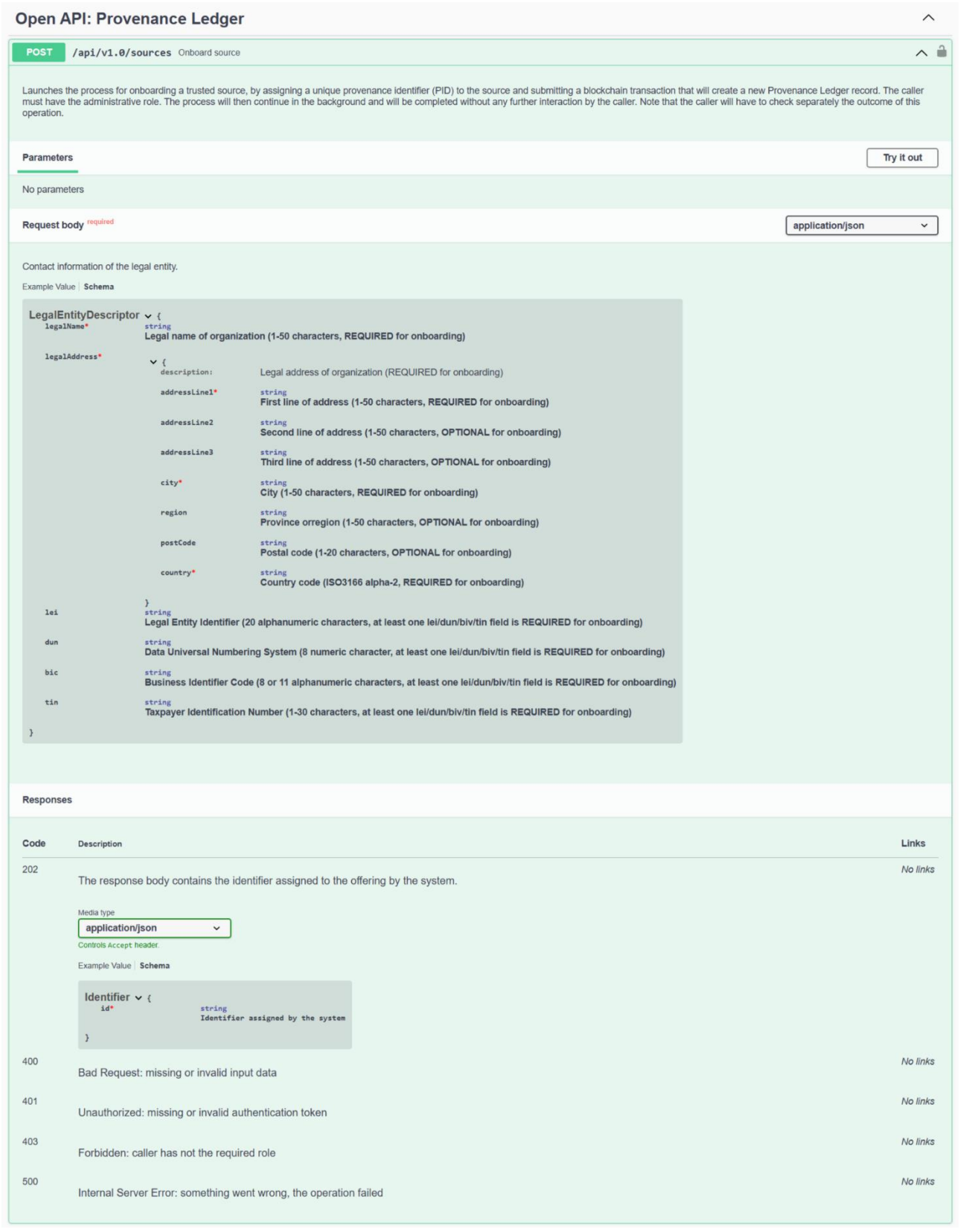

Figure 12 - Onboard Source API online documentation, details (Swagger UI)

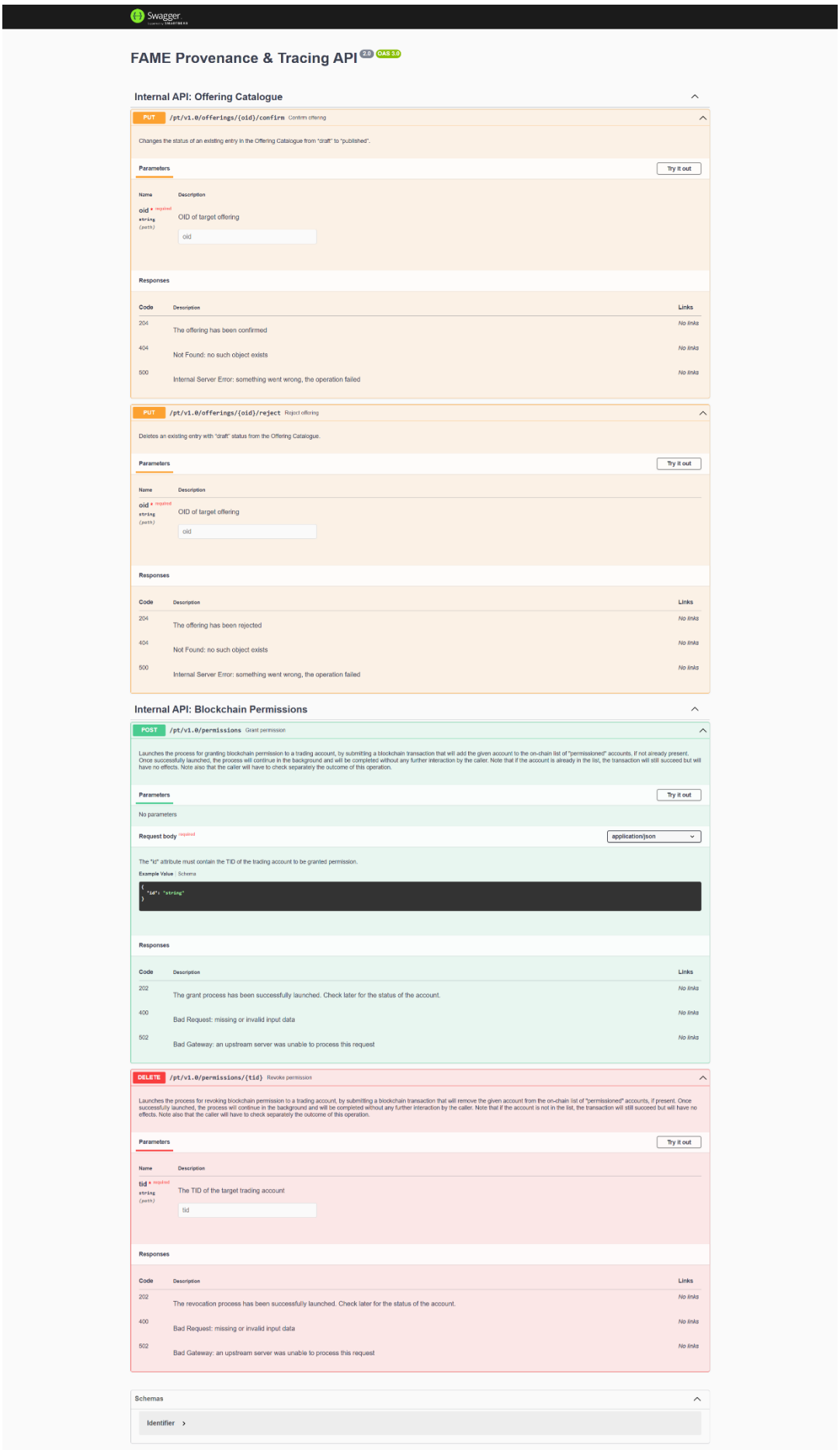

Figure 13 - P&T Internal API online documentation, overview (Swagger UI)

As already mentioned in the first release of this document, the P&T API is a modern, modular NestJS/TypeScript application that was developed following industry-standard best practices. Its source code is maintained on a private GitLab repository that is managed by the GFT project partner (see Figure 14). By the end of the project, the source code will be made publicly available under the Apache 2.0 Open-Source license.

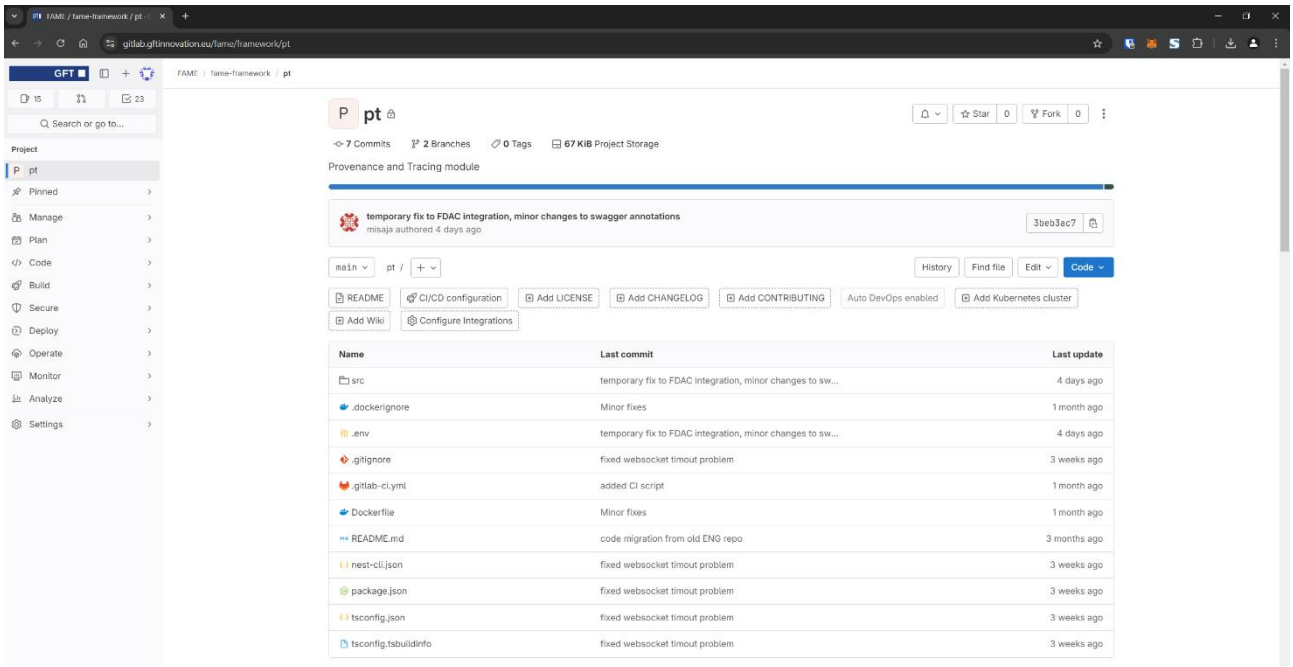

Figure 14 - The P&T code base on the FAME GitLab repository

As most other modules of the FAME Platform, P&T is distributed as a Docker container. The container image is built in such way that all the information required for integrating with other modules (e.g., the network address of service endpoints exposed by Operational Governance and Trading & Monetization) is provided at deployment time as environment variables. This mitigates cross-module dependencies and allows for the module to be easily relocated anywhere in case of need. As proof of this claim, at the end of 2024 the entire FAME Platform, including the P&T module, was migrated from its old hosting environment (a Cloud server operated by ENG) to a more scalable and robust facility based on AWS/Kubernetes (Figure 15).

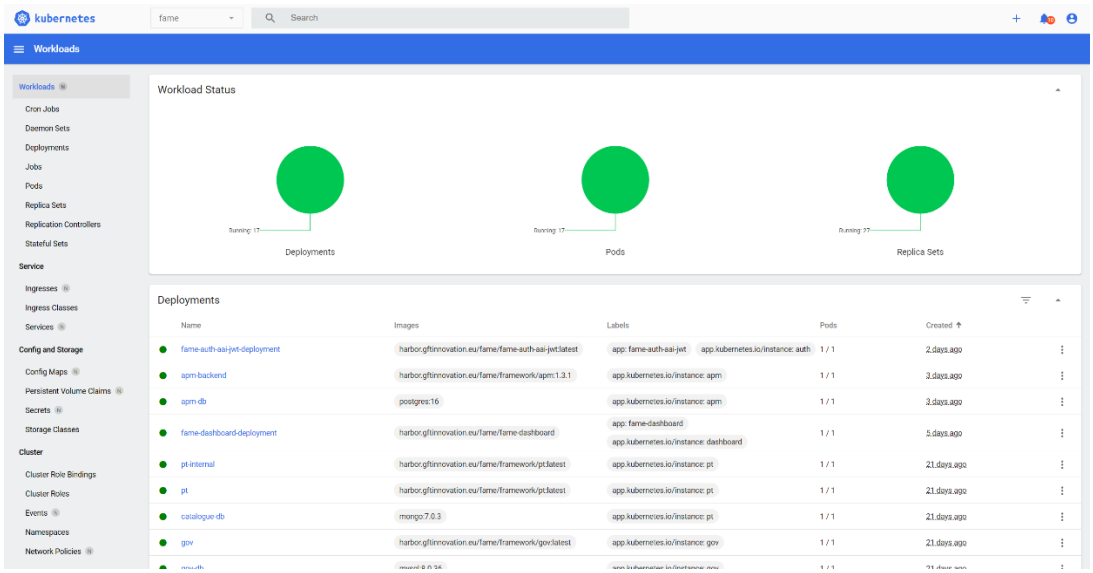

Figure 15 - FAME Platform deployment seen from the Kubernetes dashboard

5 Conclusions

With this deliverable, we concluded the FAME journey of the Provenance and Tracing module. In its current form, this software has reached a sufficient level of maturity to be used outside of the scope of the FAME project: all the required functionalities – those identified in the requirement gathering phase as well as those designed as gap-fillers during the internal experimentation done with the first release – are there, and the robustness of their implementation has been put to test, although still in a lab environment. All this is of particular importance for the FAME Platform, as the P&T module plays the role of "orchestrator" for most other modules of the FAME platform.

In the first release, the module was qualified and a *minimum viable product* (MVP), due to the lack of advances functionalities and the low *technology readiness level* (TRL) achieved at that time. This is certainly not true anymore, although there is still room for improvement. The current maturity is, according to our assessment, TRL6. We believe that it will reach TRL7 when, in the scope of task T2.3 "Data Marketplace Platform Integration", the cross-module integration will be extensively tested.

Further evolution of the module, beyond the end of the FAME project, is not only possible but likely. The P&T software is to be released under a business-friendly open-source license, and the same is true for other modules of the FAME Platform. This clearly facilitates the uptake of FAME results by other actors.