Federated decentralized trusted dAta Marketplace for Embedded finance

D4.1 - Blockchain-based Data Provenance Infrastructure I

Title	D4.1 - Blockchain-based Data Provenance Infrastructure I
Revision Number	1.0
Task reference	T4.1
Lead Beneficiary	ENG
Responsible	Mauro Isaja
Partners	FTS, INNOV, TRB, UBI
Deliverable Type	DEM
Dissemination Level	PU
Due Date	2023-12-31 [Month 12]
Delivered Date	2024-01-15
Internal Reviewers	FTS 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	2023-12-15	ENG	First version TOC and some contents
0.2	2024-01-11	ENG	Version peer reviewed FUJITSU, GFT
1.0	2024-01-15	ENG	Version for submission

Definitions

Acronyms	Definition
AAI	authentication authorization infrastructure
API	Application Programming Interface
BIC	Business Identifier Code
CDS	Copernicus Data Store
CPU	Central Processing Unit
DCAT	Data Catalog Vocabulary
DUN	Data Universal Numbering System
FAME	Federated decentralized trusted dAta Marketplace for Embedded finance
FDAC	Federated Data Assets Catalogue
ID	Identity
IDM	Identity Management
JSON	JavaScript Object Notation
LEI	Legal Entity Identifier
MVP	Minimum Viable Product|Platform
OS	Operating System
PC	Personal Computer
PIN	Personal Identification Number
RAM	Random Access Memory
REST	Representational State Transfer
RPC	Remote Procedure Call
SA	Supervisory Authority
SHA3	Secure Hash Algorithm v3
SLA	Service Level Agreement
SQL	Structured Query Language
SW	SoftWare
TIN	Taxpayer Identification Number
TR	Technical Requirement
UI	User Interface
UID	User Identifier
UML	Unified Modelling Language|Universal Markup Language
URL	Uniform Resource Locator
UUID	Universally Unique Identifier

Executive Summary

The deliverable D4.1 "Blockchain-based Data Provenance Infrastructure," presents the software prototype developed under Task 4.1, representing the first release in the FAME Platform's minimum viable product (MVP) implementation. The prototype focuses on decentralized data provenance and traceability. The D4.1 prototype has undergone testing within the MVP Platform, demonstrating the ability to publish data assets to the FAME federation's common catalogue through the Data Provenance Infrastructure.

The document outlines the logic and technical specifications of the very first release of the Provenance and Tracing (P&T) module, which implements FAME's Blockchain-based Data Provenance Infrastructure which also plays the role of "orchestrator" for other modules of the FAME platform, in the context of the Asset Publishing and Offering Definition use cases.

Key Insights:

- The prototype integrates closely with tasks and deliverables such as Federated Catalogue of Data Assets, Smart Contracts for Programmable Trading and Monetization, and Business and Operational Models for the FAME Marketplace.
- It relates to work in Platform Architecture, Data Marketplace Platform Integration, Federated AAI Infrastructure, Unified Security Policy Management, and Accounting, Trading, Pricing, and Monetization Schemes.

In particular the document provides information related to:

Module Overview: Describes the Blockchain-based Data Provenance Infrastructure (BDPI) module's role in ensuring trust within the FAME-based federation. It supports asset publishing, offering definition, and Blockchain permissions management.

Components Specification: Details the technical specifications of each module's components, including Integration Hub, Blockchain Infrastructure, Provenance Ledger, Tracing Ledger, and Offering Catalogue.

Module Demonstration: Offers instructions on setting up the prototype and provides a visual demonstration of Asset Publishing and Offering Definition workflows.

It is worth noting, the BDPI module, positioned in the FAME Solution Architecture's C4 model, plays a crucial role in maintaining the trustworthiness of published assets. The Integration Hub orchestrates cross-module interactions, supporting both public and internal API layers.

The document concludes with an Installation Guide for the 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

1 Introduction

1.1 Objective of the Deliverable

This document is a simple *factsheet* describing the software prototype released as deliverable D4.1 "Blockchain-based Data Provenance Infrastructure". 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 M24, will include additional features and will have a tighter integration with the other Platform modules.

In its current form, the D4.1 prototype has been tested and demonstrated in the scope of the MVP Platform. This partially integrated demonstrator has the capability of publishing data assets to the FAME federation's common catalogue through the Data Provenance Infrastructure and allows publishers to define commercial offerings from them.

1.2 Insights from other Tasks and Deliverables

Deliverable D4.1 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.

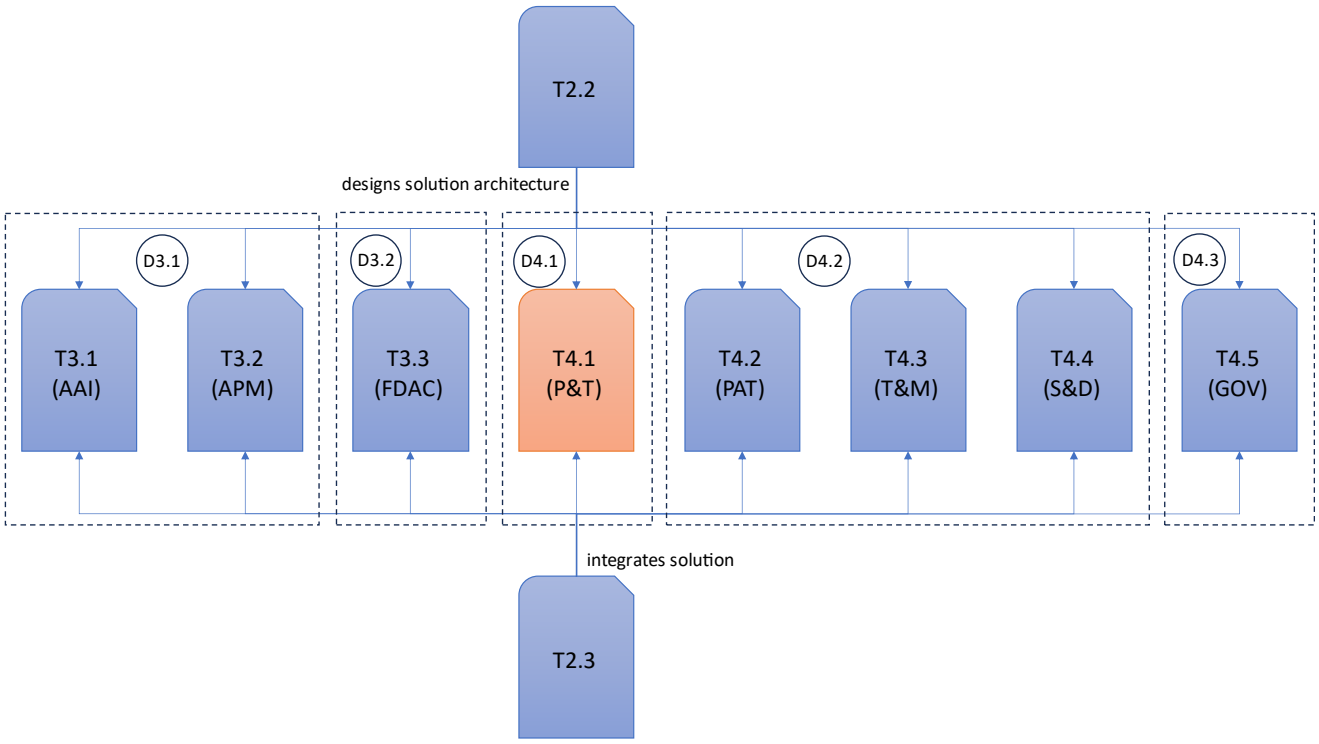

Figure 1 - Task / deliverable relationships

The above figure also introduces the shorthand notations for (most of) the software modules of the FAME Platform – e.g., AAI stands for Authentication and Authorization Infrastructure, P&T for Provenance and Tracing (the outcome of this deliverable), etc.

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 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 – Contains a recap of the achievements and the outlook for future activities.

2 Module Overview

The Blockchain-based Data Provenance Infrastructure (BDPI) module of the FAME Platform is, together with the Authentication and Authorization Infrastructure (AAI) module, one of the main *enablers of trust* of any FAME-based federation: it ensures the appropriate level of confidence in asset that are published at the federation level. In addition to this key responsibility, BDPI also supports the FAME Dashboard and FAME Open API layers in delivering specific Platform functionality to users with "Data Provider" or "FAME Administrator" role. In particular, the module enables Data Providers to publish assets' metadata and commercial offerings on the federation's common catalogue (see §3.1 "Integration Hub"), and allows FAME Administrators to manage Blockchain permissions and Blockchain-based registries (see §3.2 "Blockchain Infrastructure").

The current version of the software fully supports two key end user-oriented processes, that in this document are referred to as "Asset Publishing" and "Offering Definition" (see §2.3 "Workflows and Integration with other Modules"). Administrator-oriented processes are supported at the core level, but user interfaces are still a work in progress.

2.1 Positioning into FAME SA

In the C4 architecture model of the FAME Solution Architecture, the BDPI module is classified as a Container named "Provenance & Tracing" (P&T in brief), 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).

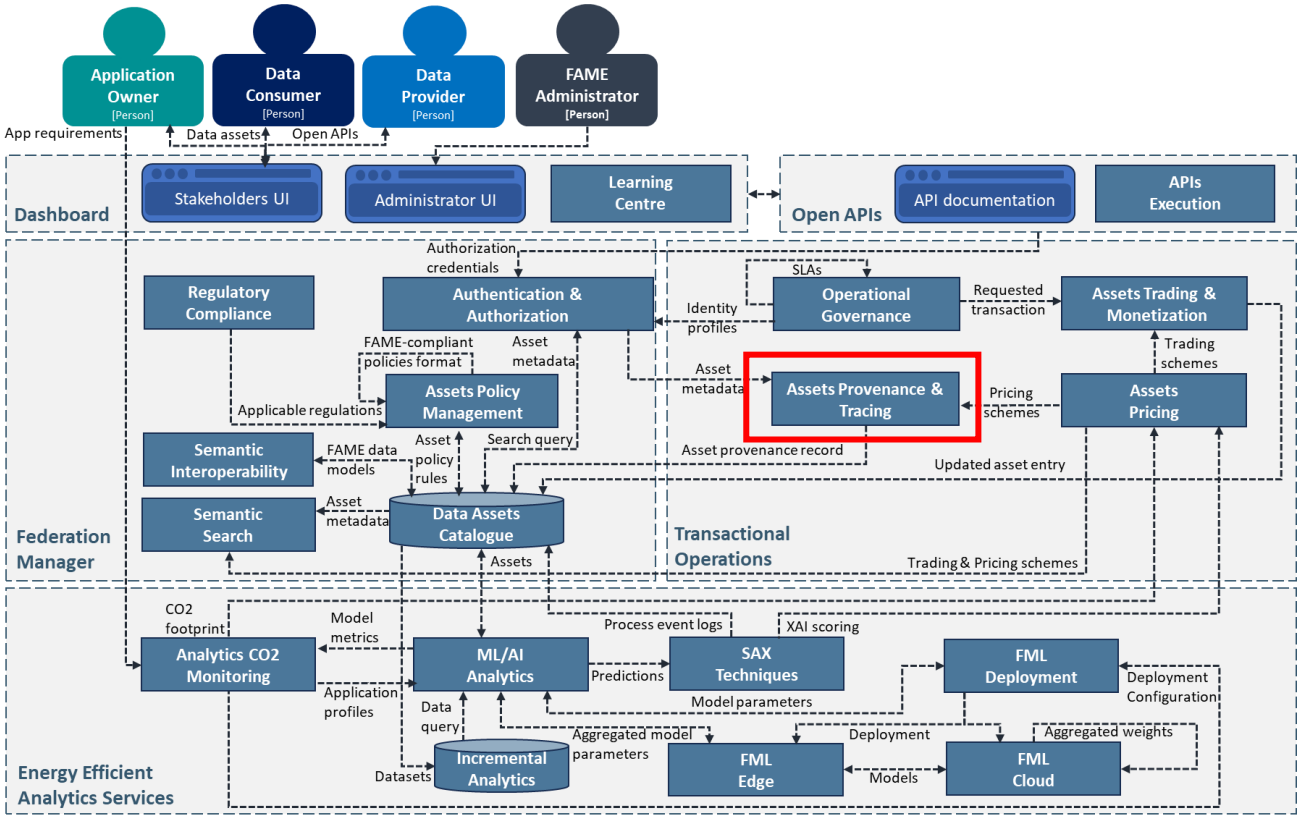

Figure 2 – FAME Solution Architecture: C4 context-level diagram (P&T Container highlighted)

2.2 C4 Component-level Architecture

In this section we provide an update to the C4 Component-level diagram of the P&T Container, as included in the D2.2 "Technical Specifications and Platform Architecture" deliverable that was released in October 2023. Such update – see Figure 3 – reflects the evolution of the FAME Platform's design during the last three months of our agile process. With respect to the previously published version, the differences are both cosmetic and substantial. The former amount to a different naming of Components, with the new names being better aligned with the Component's function; the latter include the introduction of a new Component ("Integration Hub") and the merging of two existing Components into one ("Blockchain Infrastructure Service Endpoint" and "Distributed Ledger" are now jointly represented as "Blockchain Infrastructure"). Moreover, the diagram now shows how the Integration Hub Component orchestrates all P&T's interactions with the FDAC, GOV, and T&M Containers – see §2.3 below for more details.

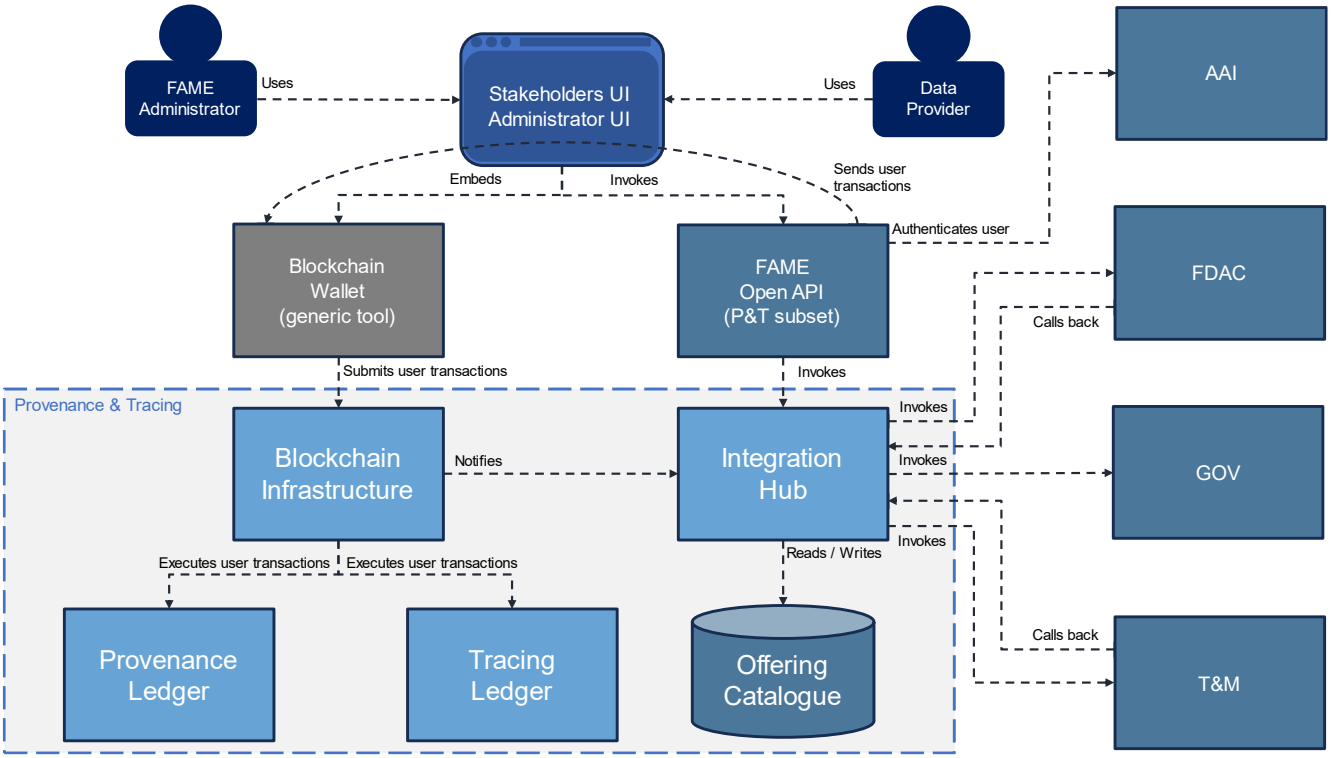

Figure 3 – P&T Container: C4 Component-level diagram

2.3 Workflows and Integration with other Modules

As implied by its name, the Integration Hub Component is where cross-module integration (for the purposes of P&T) is implemented. This Component provides two distinct API layers: *public* and *internal*. The public API provides service endpoints for user-level operations – i.e., operations that end-users execute directly through the FAME Open API layer, or indirectly through the mediation of the FAME Dashboard. The internal API, on the other hand, is only available to the other modules of the FAME Platform. Another fundamental difference between the two layers is that public API endpoints are executed in the scope of a *security context* – created by the Open API layer, thanks to is integration with the Authentication and Authorization Infrastructure (AAI) – which carries key information about the calling user: *identity*, *affiliation*, and *role*.

Here below (Figure 4 and Figure 5), the two P&T workflows that are supported by the current version of the software are documented as UML interaction diagrams, where the integration with the other Platform modules is explained in detail.

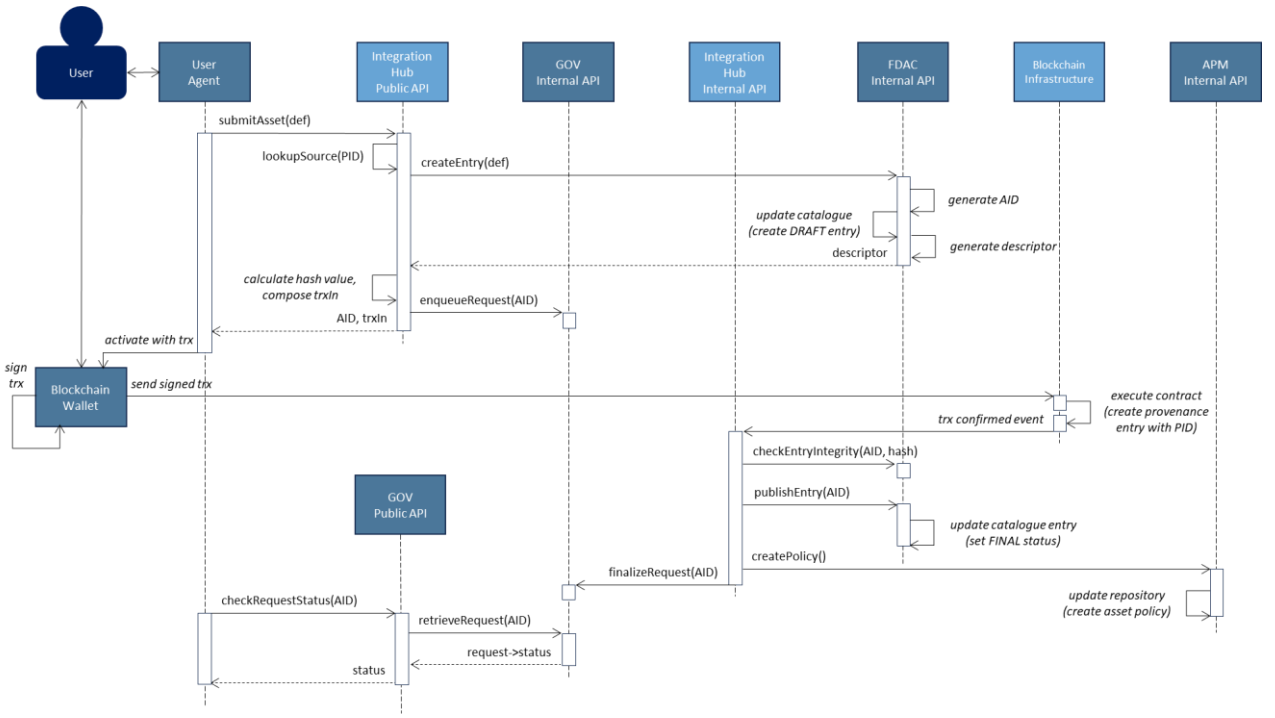

Figure 4 - Asset Publishing workflow

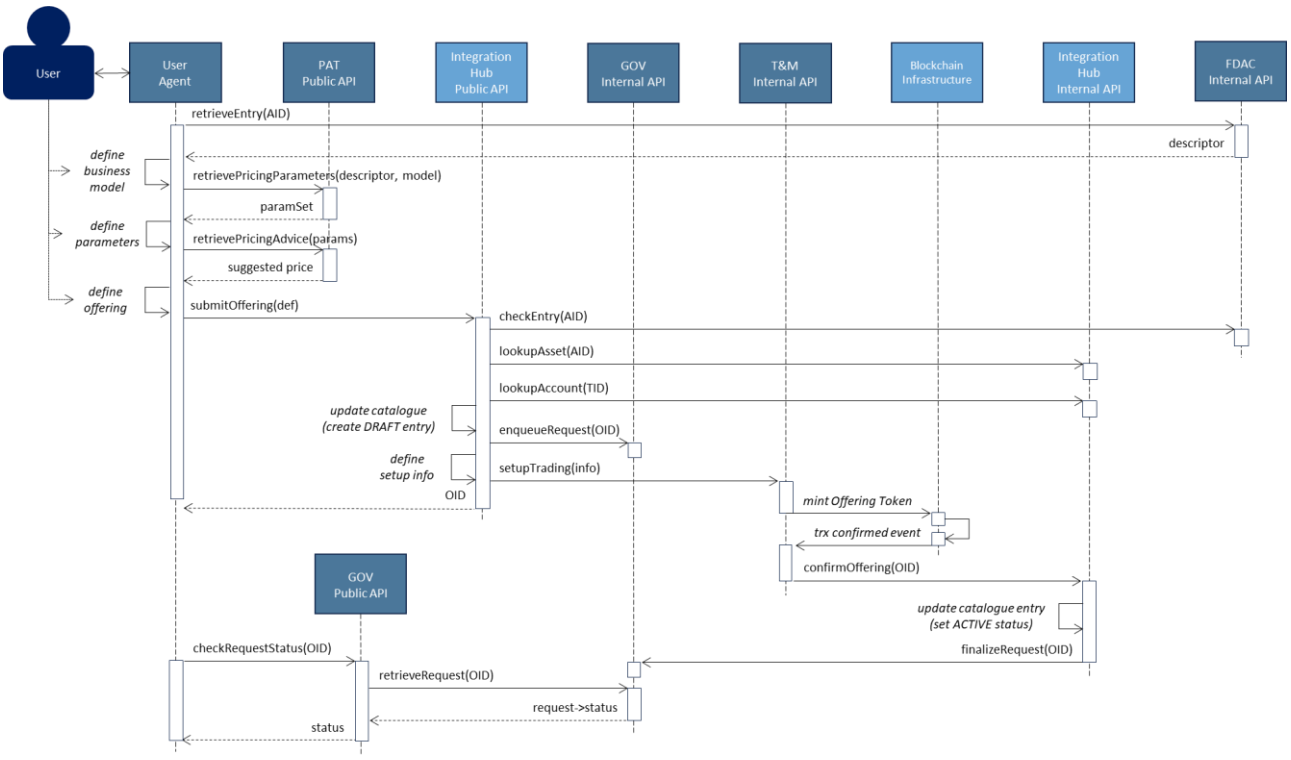

Figure 5 - Offering Definition workflow

When an API call triggers a blockchain transaction, the business logic is executed asynchronously: the call immediately returns a Request ID (RID), so that the caller can check the status of their pending request at any time with the GOV module. This is the case in the Offering Definition workflow described above (Figure 5): once the internal call `setupTradingInfo` is dispatched by the Integration Hub Component to the T&M module, a Blockchain transaction that is signed by a system-owned account is processed in the background; then, if and when such transaction is successful, all the required updates to the Offering Catalogue and the GOV module will be applied.

The Asset Publishing workflow (Figure 4) follows a similar approach, but with a big difference: the Blockchain transactions must be submitted directly by the calling user, because a new record in the Tracing Ledger must be created *on behalf of that user*. In order to meet this nontrivial requirement, the user must have a MetaMask wallet plug-in installed in their web browser (also known as the "trading wallet", see §4.1.3), must own a Blockchain account and must undergo a registration process that grants transaction permission to that account (see the "Enable account" operation of the Integration Hub Component - §3.1.2.2).

3 Components Specification

In the following sections, the five Components within the P&T Container are analysed individually.

3.1 Integration Hub

3.1.1 Description

The Integration Hub, as the name suggests, is tasked with the coordination of workflows that involve one *or more* modules of the FAME Platform. It provides the API layer of the P&T Container, with both public and internal service endpoints (see §2.3), which are documented in §3.1.2.2. This is the only Component of the P&T Container exposing an interface: all the others are for internal use.

3.1.2 Technical Specification

3.1.2.1 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

3.1.2.2 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*.

- **Submit asset [public]**
  Receives the description of an asset, which is formatted as an Asset Descriptor object (see §3.1.2.3) that does *not* include the AID (asset identifier) attribute. The service instructs the FDAC module to create a temporary entry: the FDAC will assign a unique AID and DRAFT status to the asset and return an Asset Descriptor object that includes the AID attribute. The service then calculates the hash value of the Asset Descriptor object and prepares the input data – that includes the AID and the calculated hash value – for a blockchain transaction that targets the Tracing Ledger smart contract (see §3.4). Finally, the service instructs the GOV module to append a new entry – identified by the AID and with PENDING status – to the request queue and returns both the AID and the transaction's input data to the caller.

```
POST https://<public_address>/api/v1.0/submissions/assets
(affiliation PID retrieved from security context)
Request body (application/json)
{
  (Asset Descriptor object without the AID attribute)
}
```

```
Response body (application/json)
{
  "rid": ".." /* AID, which also identifies the caller's request */
}
```

Note that, at this point, the asset is *not* yet published. The caller is expected to sign and submit the received blockchain transaction, using their own trading wallet and the credentials of their registered blockchain account (see the *Enable account* operation). When the Integration Hub will receive from the Blockchain Infrastructure (see §3.2) an internal *transaction approval notification*, it will proceed as follows to complete the publishing process:

- instruct the FDAC module to promote the entry to FINAL status;
- instruct the APM module to create a new policy for the published asset;
- instruct the GOV module to set the status of the original request to PROCESSED.

- Lookup asset [internal]
  Receives an AID. If a matching item in the Tracing Ledger exists, returns the complete ledger record, which includes the publisher's PID and the hash value of the published metadata.

```
GET http://<private_address>/pt/v1.0/assets/{aid}
Response body (application/json)
{
  "aid": "..",       /* asset ID */
  "pid": "..",       /* provenance ID (registered publisher) */
  "published": "..", /* timestamp of publishing (ISO8601) */
  "hash": ".."       /* hash value of Asset Descriptor */
}
```

- **Submit offering [public]**
  Receives the description of a commercial offering, which is formatted as an Offering Descriptor object (see §3.1.2.3), for an already published asset. Such definition includes the TID (trading account identifier) of the blockchain account designated as the beneficiary in trading transactions, and program-level instructions to the CDS (Content Delivery Service) that is responsible for serving digital content to the buyers. The service creates a temporary entry in the Offering Catalogue (see §3.5) with DRAFT status. Such entry is identified by an OID (offering identifier) consisting of the hash value of the Offering Descriptor object, calculated using the SHA3-256 algorithm and formatted through the Base58Check algorithm[1]. The service then instructs the GOV module to append a new entry – identified by the OID and with PENDING status – to the request queue and instructs the T&M module to setup the trading environment for the offering. Finally, it returns the generated OID to the caller.

```
POST https://<public_address>/api/v1.0/submissions/offerings
(affiliation PID retrieved from security context)
Request body (application/json)
{
  "offering": {..}, /* Offering Descriptor object */
  "tid": "..",      /* beneficiary account */
  "target": {..},   /* instructions to CDS: content specs */
  "sl": {..}        /* instructions to CDS: service level specs */
}
```

[1] 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.

```
Response body (application/json)
{
  "rid": ".."  /* OID, which also identifies the caller's request */
}
```

Note that, at this point, the offering is *not* yet active: the T&M module will call the *Confirm offering* service endpoint (see below) to confirm that the trading environment has been initialized, thus activating the offering. However, the T&M module may instead call the *Reject offering* service endpoint, to the effect that the submitted offering is discarded.

- Confirm offering [internal]
  Receives an OID. If the OID corresponds to an entry in the Offering Catalogue that has DRAFT status, the service sets the status of the entry to ACTIVE and instructs the GOV module to set the status of the original request (matched by OID) to PROCESSED.

```
PUT http://<private_address>/pt/v1.0/offerings/{oid}/confirm
```

- Reject offering [internal]
  Receives an OID and an error description. If the OID corresponds to an entry in the Offering Catalogue that has DRAFT status, the service deletes the entry and instructs the GOV module to set the status of the original request (matched by OID) to ERROR, appending the error description to the request.

```
PUT http://<private_address>/pt/v1.0/offerings/{oid}/reject
```

- Retrieve offering [internal]
  Receives an OID. If the OID corresponds to an entry in the Offering Catalogue, returns the matching entry – which includes the Offering Descriptor and the complete meta-data (status, etc.).

```
GET http://<private_address>/pt/v1.0/offerings/{oid}
Response body (application/json)
{
  "offering": "..",     /* Offering Descriptor object */
  "status": "..",       /* current status
                           domain: DRAFT | ACTIVE | ARCHIVED */
  "lastUpdated": "..",  /* timestamp of last update (ISO8601) */
  "lastUpdatedBy": ".." /* UID of last update */
}
```

- **Retrieve active offering [public]**
  Receives an OID. If the OID corresponds to an entry in the Offering Catalogue that has ACTIVE status, returns the matching Offering Descriptor object.

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

- **List active offerings [public]**
  Receives an AID (asset identifier). Returns a pageable list of OIDs, each OID corresponding to an entry in the Offering Catalogue that is linked to the given asset and has ACTIVE status.

```
GET https://<public_address>/api/v1.0/offerings?aid={aid} [&l={n}&o={n}]
Response body (application/json)
["..",".."]
```

- Enable account [internal]
  Receives a TID (trading account identifier) and, optionally, a PID (provenance identifier). If the TID is not already authorized on the Blockchain Infrastructure (see §3.2), adds the TID to the list of authorized accounts. If a PID argument is provided, adds the PID-TID association to the Provenance Ledger (see §3.3)[2].

```
POST http://<private_address>/pt/v1.0/accounts
Request body (application/json)
{
  "tid": "..", /* trading account REQUIRED */
  "pid": ".."  /* registered legal entity that owns the account */
}
```

- Lookup account [internal]
  Receives a TID. Reads the PID corresponding to the TID, if any, from the Provenance Ledger, and returns the result to the caller.

```
GET http://<private_address>/pt/v1.0/accounts/{tid}
Response body (application/json)
{
  "pid": ".."  /* registered legal entity that owns the account */
}
```

- Register source [internal]
  Receives the description of a legal entity. Generates a unique provenance identifier (PID) and adds the PID and the legal entity description to the Provenance Ledger. Returns the PID.

```
POST http://<private_address>/pt/v1.0/sources
Request body (application/json)
{
  (Legal Entity Descriptor object)
}
Response body (application/json)
{
  "pid": ".."  /* UUID assigned by system */
}
```

- Lookup source [internal]
  Receives a PID. If the PID identifies an *active* item in the Provenance Ledger, returns the description of the legal entity.

```
GET http://<private_address>/pt/v1.0/sources/{pid}
```

[2] The PID must be already registered in the Provenance Ledger, and the TID must not be already associated with any PID, or an error will occur.

```
Response body (application/json)
{
  (Legal Entity Descriptor object)
}
```

3.1.2.3 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.

```
{
  /* THE UNIQUE ID IS ASSIGNED BY THE SYSTEM */
  "dcterms_identifier": "..",         /* asset ID (AID) */
  /* ALL THE REMAINING ATTRIBUTES ARE ASSIGNED BY THE PUBLISHER */
  "dcterms_title": "..",              /* name of asset REQUIRED */
  "dcterms_description_short": "..",  /* short description of asset REQUIRED */
  "dcterms_type": "..",               /* content type REQUIRED
    domain: DATASET | DATASTREAM | MODEL | DOCUMENT | MEDIA | SOFTWARE | SERVICE */
  "dcat_endpointURL": "..",           /* content server (URL) REQUIRED */
  "dcat_landingPage": "..",           /* descriptive/support web page (URL) */
  "dcat_keyword": ["..", ".."],       /* keywords attached to asset */
  "dcterms_conformsTo": ["..", ".."], /* standards the asset contents conform to (URLs) */
  "dcterms_description_long": "..",   /* long description of asset */
  "fame_logo": ".."                   /* logo of asset (URL) */
}
```

- Offering Descriptor
  JSON literal that describes a commercial offering for a published asset.

```
{
  "asset": "..",   /* link to 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: nH | nD | nW | nM | nY */
    "downloads": ..       /* maximum number of complete downloads: n */
    "volume": ".."        /* maximum number of downloaded bytes: nKB | nMB | nGB | nTB | .. */
  },
  "license": "..", /* formal description of terms and conditions: human readable text */
  "sla":            /* formal description of service level: human readable text */
}
```

Only the following elements are mandatory: `asset`, `title`, `price`, `summary`, `license`. The elements contained in the `cap` section are mutually exclusive: when the `cap` section is present, it can only contain one element.

- 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.

```
{
  /* REQUIRED: only one of LEI, DUN, BIC, TIN */
  "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": "..",      /* REQUIRED */
  "LegalAddress": {
    /* only AddressLine1, City and Country are required */
    "AddressLine1": "..", /* REQUIRED */
    "AddressLine2": "..",
    "AddressLine3": "..",
    "City": "..",         /* REQUIRED */
    "Region": "..",
    "Postcode": "..",
    "Country": ".."       /* ISO3166 alpha-2 code REQUIRED */
  }
}
```

3.2 Blockchain Infrastructure

3.2.1 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.

3.2.2 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 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

3.2.2.2 Deployment

Presently, only one RPC service endpoint is available to users. This is currently deployed at this network address, which is reachable from the public Internet: http://162.55.94.15:8545

There is also a Quorum Explorer instance that can be reached at this address: http://162.55.94.15:25000/explorer/nodes. Note however that user credentials are required for access, and these are only released to administrators.

3.3 Provenance Ledger

3.3.1 Description

The Provenance Ledger keeps record of the unique identifier and full contact details of *legal entities* that have been registered as *verified content publishers*. This registry, implemented as a blockchain smart contract deployed on the Blockchain Infrastructure (see §3.2), enables the assignment of an unambiguous and trustworthy *provenance identifier* (PID) to all digital assets that are published on the FAME federation. Additionally, the registry links publishers to their trading accounts (TID), to the effect that the authenticity of content publishing transactions can be enforced by the P&T module.

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 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

3.3.2.2 Data Structures

Legal Entity Record

- **PID** (unique ID of organization: UUID)
- **Descriptor** (Legal Entity Descriptor)
- **Created** (time of creation: timestamp)
- **Archived** (time of archival, if ever: timestamp)
- **Status** (ACTIVE | ARCHIVED)

Each record has 0-N associated TID (trading account identifier) entries, to track the ownership of TIDs.

3.4 Tracing Ledger

3.4.1 Description

The Tracing Ledger is a smart contract, deployed on the Blockchain Infrastructure (see §3.2), 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 §3.3). 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.

3.4.2 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 4 – Tracing Ledger's baseline technologies

Baseline Technology	Description	Added value to FAME
Hyperledger Besu's Ethereum Virtual Machine	Runtime environment for Solidity-based smart contracts	Implementation of the Tracing Ledger's persistence layer

3.4.2.2 Data Structures

Asset Record

- **Id** (asset identifier - AID)
- **Hash** (hash value of the asset metadata, as published on the FDAC module)
- **provenanceId** (publisher identifier - PID)
- **timestamp** (date/time of creation)

3.5 Offering Catalogue

3.5.1 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.

3.5.2 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 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

3.5.2.2 Data Structures

Offering Record

- **OID** (offering identifier - OID)
- **Asset** (asset identifier - AID)[3]
- **Descriptor** (Offering Descriptor object)
- **Created** (date/time of creation)
- **Created by** (user who submitted the offering: UID)
- **Archived** (date/time of archival, if any)
- **Archived by** (user who archived the offering: UID)
- **Status** (DRAFT | ACTIVE | ARCHIVED)

[3] This information is included in the Offering Descriptor but is replicated as an indexable field to speed-up queries.

4 Module Demonstration

This section has two purposes: firstly, to provide the reader with the instructions for retrieving and installing the complete software implementation of the Provenance and Tracing module of the FAME Platform – i.e., the five Components described in §3; secondly, walking the reader through a demonstrator that implements the "minimum viable product" configuration of the *integrated* FAME Platform – i.e., the Asset Publishing and Offering Definition use cases.

Regarding the integrated demonstrator, we have to point out that it relies on several modules that are *not* in the scope of this deliverable: Pricing Advisory Tool (PAT), Trading and Monetization (T&M), Federated Data Asset Catalogue (FDAC), Asset Policy Management (APM), Operational Governance (GOV), Authentication & Authorization Infrastructure (AAI). The demonstrator adopts a mixed approach to deployment: most of the software is installed on one single computer, but the FDAC and AAI modules are only available as a service and thus are hosted by their respective providers. Figure 6 shows how this kind of deployment looks like: the Main Infrastructure and the User's PC are the two target environments of the Installation Guide (see §4.1), while the Catalogue Infrastructure and the Authentication Service Provider are two external systems. With respect to this, it is important to note that both of these external systems must also be configured for the specific installation and must be operational in order for the demonstrator to work: we recommend that the reader gets in touch with the FAME teams that are responsible for these modules[4] before planning any local installation.

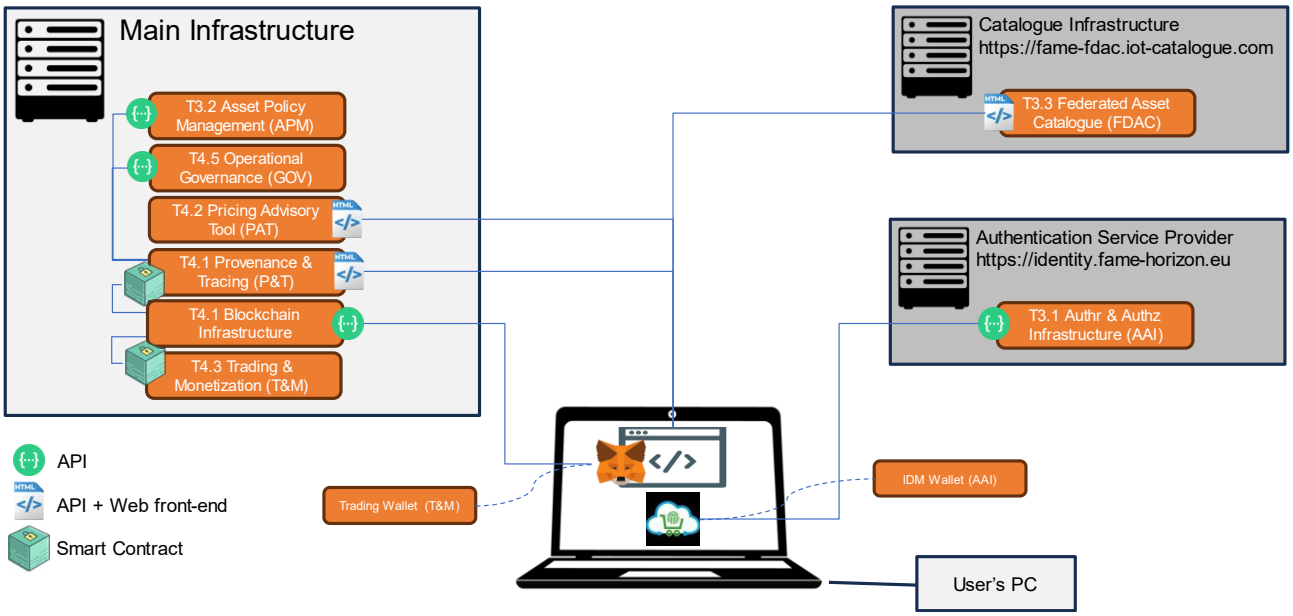

Figure 6 - Demonstrator deployment

4.1 Installation Guide

4.1.1 Prerequisites

The Main Infrastructure environment (see Figure 6 above) should be a network server running on a x86_64 architecture with the Linux operating system, the Ubuntu distribution being the recommended version. It may be theoretically possible to install the software on different systems *that are supported by Docker*, but this is not recommended as they have not been tested. As for hardware sizing, we

[4] FDAC is managed by UNPARALLEL, AAI by NUIG.

recommend a physical server with at least one multicore CPU and 32GB of RAM, due to the need of running several Docker containers at once.

On the other hand, the User's PC can be any low-end Windows system – typically, an office laptop – having the latest version of the Chrome or Firefox web browser installed.

Both computers must be connected to the public Internet. If the server is deployed behind a firewall, the network ports that must be allowed to receive *incoming* connections are the following: 80, 443, 4203, 7005, 7006, 7007, 8080, 8180, 8545, 8546, 25000. Moreover, the server itself must be able to reach out to any Internet address/port combination without limitations. Take notice of the public Internet address of the server because it will be needed later to complete the configuration of both client and server software.

4.1.2 Setting up the Main Infrastructure Server

We assume that the reader has root or SU (super-user) access to the Linux server, either locally or through a remote terminal. The steps that are described below shall be followed in the same order as they appear.

4.1.2.1 Blockchain Infrastructure

The first step is the installation of the Hyperledger Besu blockchain platform. As part of the "quick start" installation procedure, which is documented at https://besu.hyperledger.org/private-networks/tutorials/quickstart, some additional open-source software packages will also be installed. Some of these (namely, Docker, Docker Compose and Node.js) will also form the baseline environment for the deployment of the "non-blockchain" part of the Main Infrastructure, which is described in §4.1.2.3.

4.1.2.2 Provenance Ledger and Tracing Ledger

These components are Solidity smart contracts, each of which is distributed from its own GitLab repository:
Provenance Ledger - https://gitlab.infinitech-h2020.eu/fame/framework/pt/provenance-ledger
Tracing Ledger - https://gitlab.infinitech-h2020.eu/fame/framework/pt/tracing-ledger.
Note that access permission is restricted by the GitLab instance. To receive access credentials, please send your request to the FAME coordination team.

To deploy the two smart contracts on the Besu network, go through these steps for each of them:

- Clone the distribution repository on the server, using any Git client (a command-line tool is included in most Linux distributions – if not, consult the documentation of your OS). The destination directory can be anywhere on the server's filesystem, *provided the two distributions are kept separate*.
- In the root folder of the smart contract's local copy, edit the file `truffle-config.js`: set the network address attribute of the `privateKeyProvider` configuration object to the value that is appropriate for your system – in the template below, replace the `<server_address>` placeholder with the actual address of the Main Infrastructure server (don't change the port number):

```
const privateKeyProvider =
    new PrivateKeyProvider(privateKeys, '<server_address>:8545', 0, 3);
```

- From the same root folder, execute the script `deploy.sh`.
- Examine the execution output displayed in the console: if the execution terminated without errors, take notice of the values assigned to the `CONTRACT_ADDRESS_PROVENANCE` (when installing the Provenance Ledger smart contract) and `CONTRACT_ADDRESS_TRACING` (when

installing the Tracing Ledger one) variables[5], as these "contract addresses" will be needed later on (see §4.1.2.3 below).

4.1.2.3 Non-blockchain Components

All the remaining software that must run on the Main Infrastructure's server has been packaged into a set of Docker images, which can be easily deployed as a single unit, thanks to Docker Compose automation. The package is distributed from the same GitLab facility mentioned in §4.1.2.2 (the same access restrictions apply), and is available at this path: https://gitlab.infinitech-h2020.eu/fame/framework/demo/-/archive/main/integration-main.zip.

To deploy and launch all the Docker containers, go through these steps:

- Download the distribution file on the server and unpack it anywhere on the server's filesystem.
- In the root folder of the unpacked distribution, edit the file `docker-compose-pt.yaml`:
  - set the `rpc-node` attribute with the value that is appropriate for your system (e.g., `rpc-node:<server_address>`);
  - set the `CONTRACT_ADDRESS_PROVENANCE` attribute with the value assigned by the Provenance Ledger smart contract's deployment procedure (see §4.1.2.2);
  - repeat the previous step for the `CONTRACT_ADDRESS_TRACING` attribute, setting the smart contract address of the Tracing Ledger's deployment.
- From the same root folder, execute the script `setup.sh`.

4.1.3 Setting up the User's PC

Setting up the User's PC is a matter of installing two separate wallets: one for identity management (IDM) and one for trading. Both are externally developed software that is just *used* in the context of the FAME demonstrator.

The trading wallet is a very well-known and mature browser extension, MetaMask, that enables any supported browser to interact with Ethereum-compatible blockchain networks – in our case, FAME's Blockchain Infrastructure (§3.2) – by signing and submitting transactions. Installing the MetaMask extension is trivial: just open the web browser, navigate to https://metamask.io/ and follow the instructions provided by the web site. As part of the installation process, you should create your own local wallet, encrypted with a – possibly strong – password (Figure 7 - MetaMask extension: creating a password-protected wallet). Once this step is completed, the software automatically creates a new *trading account* with the default name "Account 1", a random *address* and a matching private key which is securely stored in the wallet. You must take notice of the trading account's address because it will be needed later: from the MetaMask menu, select "Account details" and copy the address to the clipboard (Figure 8). You must then instruct the MetaMask extension on how to reach the FAME Blockchain, as it now points, by default, to the Ethereum public network: from the menu, select Settings/Networks, click on "Add network", follow the "Add a network manually" link and fill-in the resulting form as shown in Figure 9 – obviously, replacing the `<server_address>` placeholder with the actual address of the Main Infrastructure server (don't change the port number). You then tell MetaMask to *switch* to the FAME network: the extension is now ready to play the role of *trading wallet* and will be automatically activated when needed.

.

[5] They should look like this example: `0xfeae27388A65eE984F452f86efFEd42AaBD438FD`.

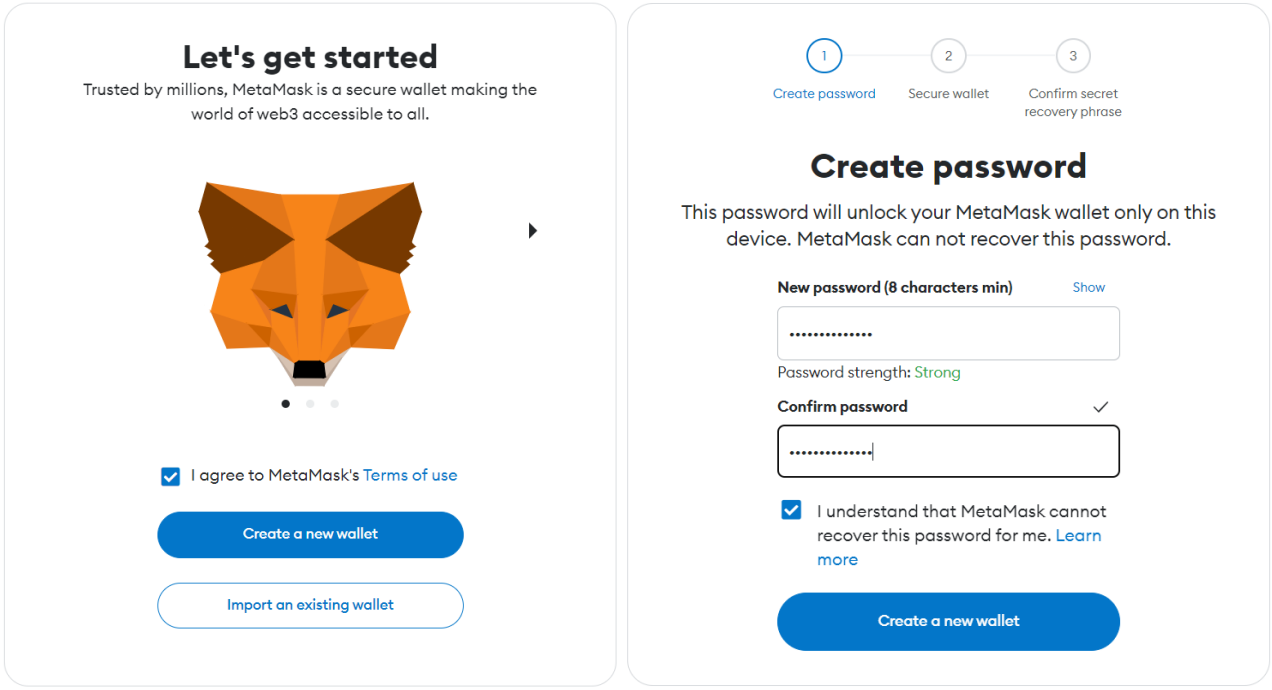

Figure 7 - MetaMask extension: creating a password-protected wallet

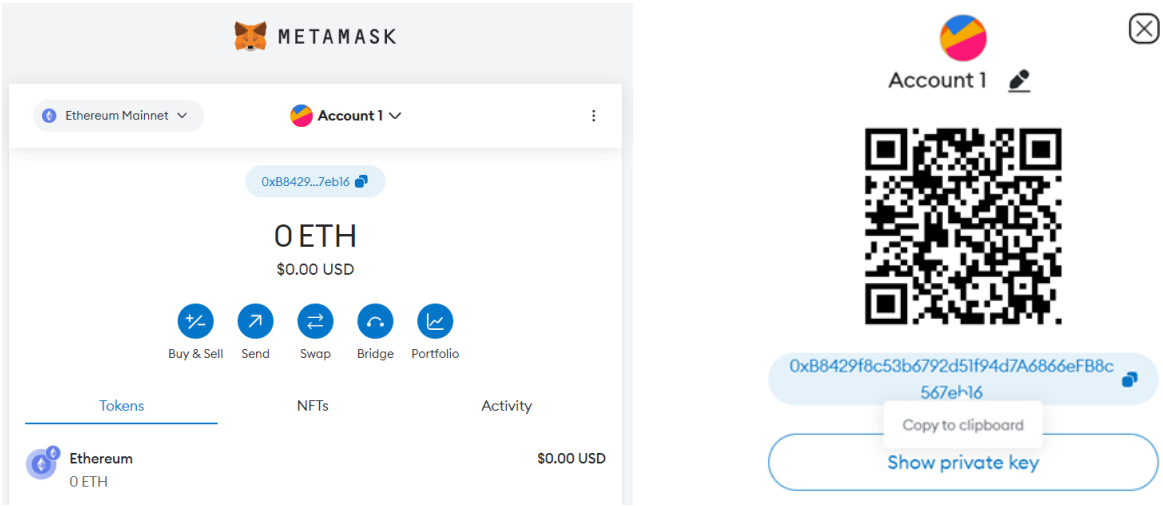

Figure 8 - MetaMask extension: retrieving the trading account's address

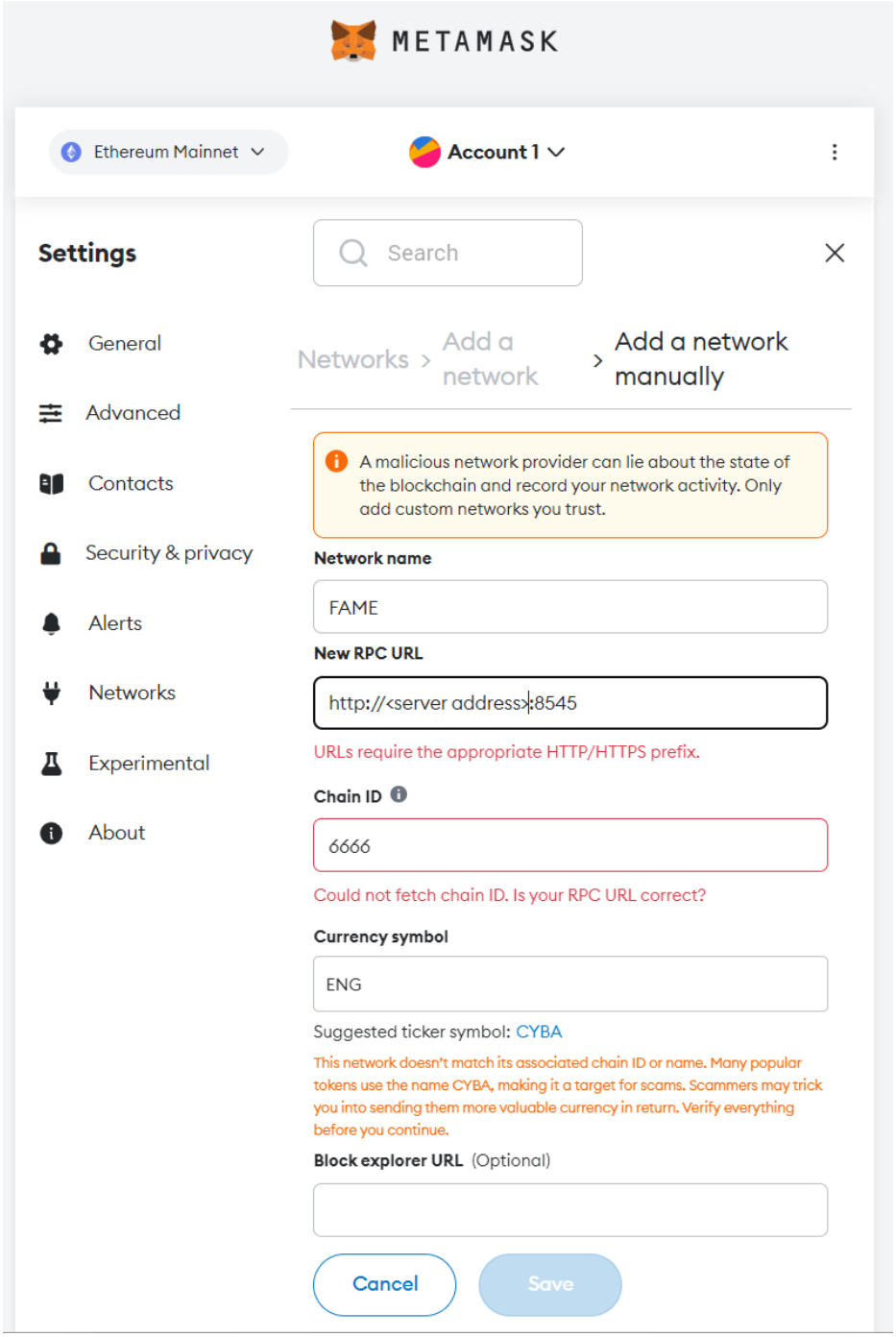

Figure 9 - MetaMask extension: connecting the FAME blockchain

The IDM wallet is, at this stage of development of the FAME Platform, a desktop application for Windows that comes from the i3-MARKET research project (https://www.i3-market.eu/)[6]. Installing the i3-MARKET IDM wallet is simple: download the installer from the public repository https://github.com/i3-Market-V3-Public-Repository/SP3-SCGBSSW-I3mWalletMonorepo/releases/download/v2.6.2/wallet-desktop-v2.6.2-x64.exe[7] and run it[8].When launched for the first time, the app will ask you to set a password. It will then display a blank startup

[6] A FAME-native replacement, with support for mobile devices, is scheduled for the next release.
[7] You need to have a valid account on GitHub to download the installer.
[8] Once launched, you may find that Microsoft Defender SmartScreen (often installed in corporate-managed PCs) is blocking the execution of the unknown application. If that is the case, you can override the block by clicking *more info / run anyway* on the pop-up window.

screen, as shown in Figure 10. You then click on the "+" button to create a new local wallet. The wallet creation wizard will go through three steps:

- Assigning a name: type "FAME"
- Choosing a type: select "HD SW Wallet"
- Choosing a blockchain network: select "i3m"

Once the FAME wallet appears in the list, you right-click on it to create a new identity (Figure 11). The name assigned to this identity is, for the purpose of running the demonstrator (see §4.2), "provider". The name, however, is just a mnemonic aid for the user who owns the identity: the true identifier is a DID[9] that is generated automatically and published on the blockchain network the wallet is connected to – i.e., i3m, selected at the third step[10]. These details are visible in the right-hand pane of the UI, when the "provider" identity is selected (Figure 12).

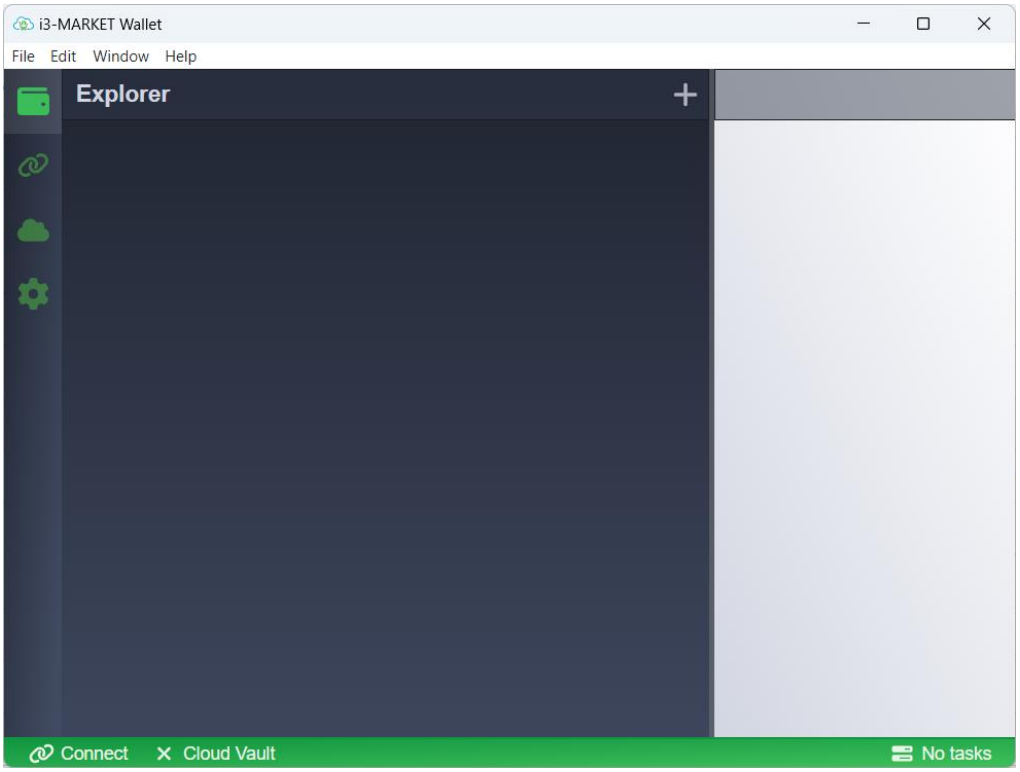

Figure 10 - i3M application: startup screen

[9] Decentralized Identifier, see https://www.w3.org/TR/did-core/

[10] Notice that the i3M wallet implementation does not use, at this stage, the FAME Blockchain Infrastructure for DID management. This will be corrected in the future.

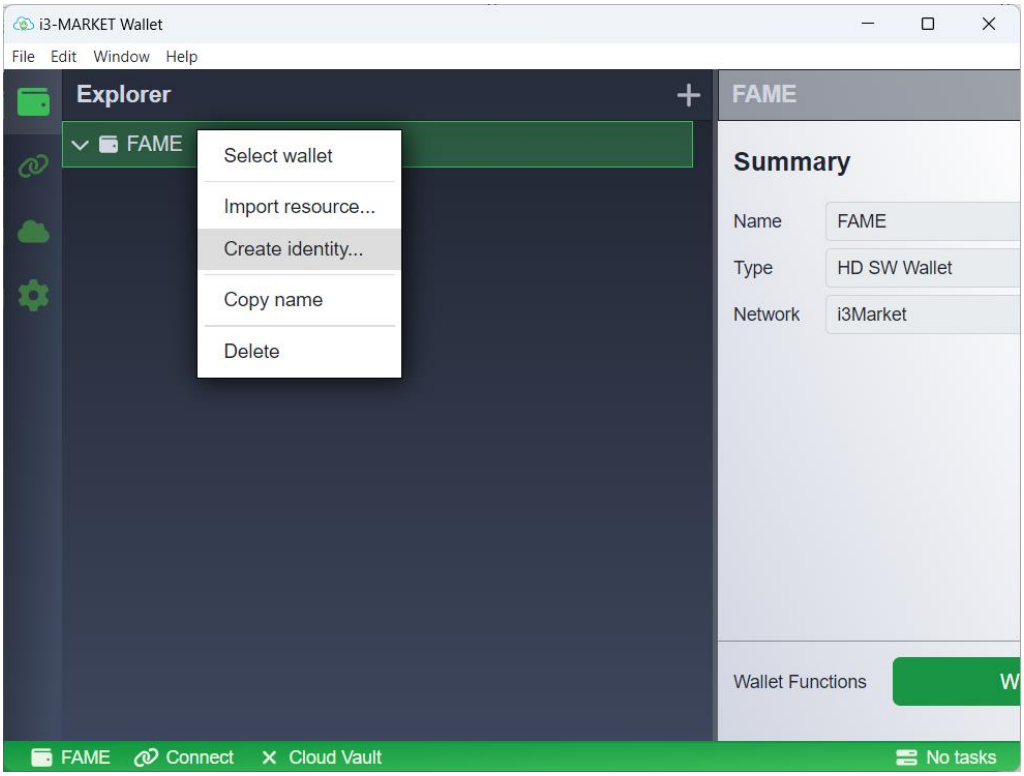

Figure 11 - i3M application: creating an identity

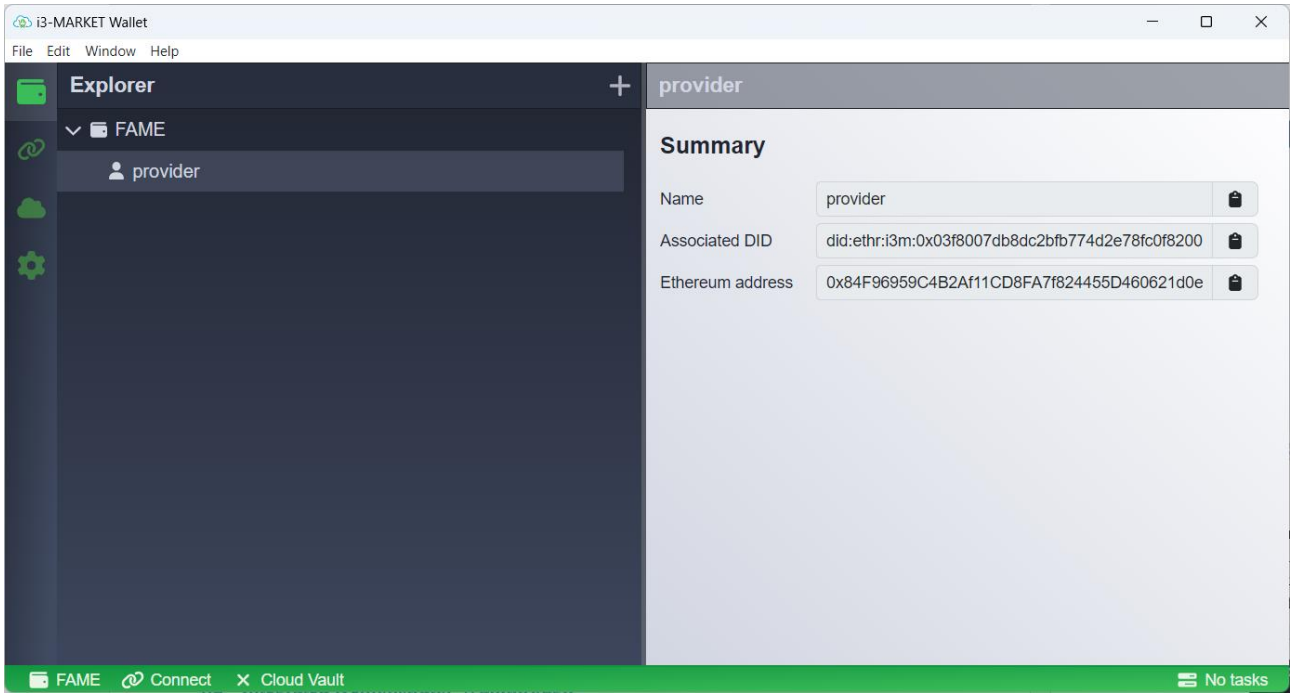

Figure 12 - i3M application: identity details

Up to this point, the "provider" identity is just a generic way of *authenticating* its owner. In order for the owner to use any of the functionalities provided by the FAME platform, however, such identity must be first *onboarded* in the FAME ecosystem. The onboarding process is not documented here, because it is out of the scope of the P&T module: onboarding and its enabling software belong to the GOV module and will be addressed in deliverable D4.3 "Operational Models, Business Models and Governance", of future release. Suffice to say that a user must receive a *Verifiable Onboarding Credential* (VOC) from a trusted *Onboarding Authority*. A VOC is just a special case of Verifiable

Credential[11], issued by some entity – the Onboarding Authority – that is registered in FAME as a trusted party. For the purpose of running the demonstrator, a software tool that *simulates* the onboarding process is included in the Main Infrastructure server package (see §4.1.2.3) – once installed, it is available here: http://<server_address>:4203/pages/credential.

4.2 Integrated Demonstrator Walkthrough

The demonstrator's walkthrough starts, as usual, with the login procedure: the User points the web browser of their PC to the URL of the Asset Publishing page (http://<server_address>:4203/pages/publish-content) and is redirected to the login page (Figure 13).

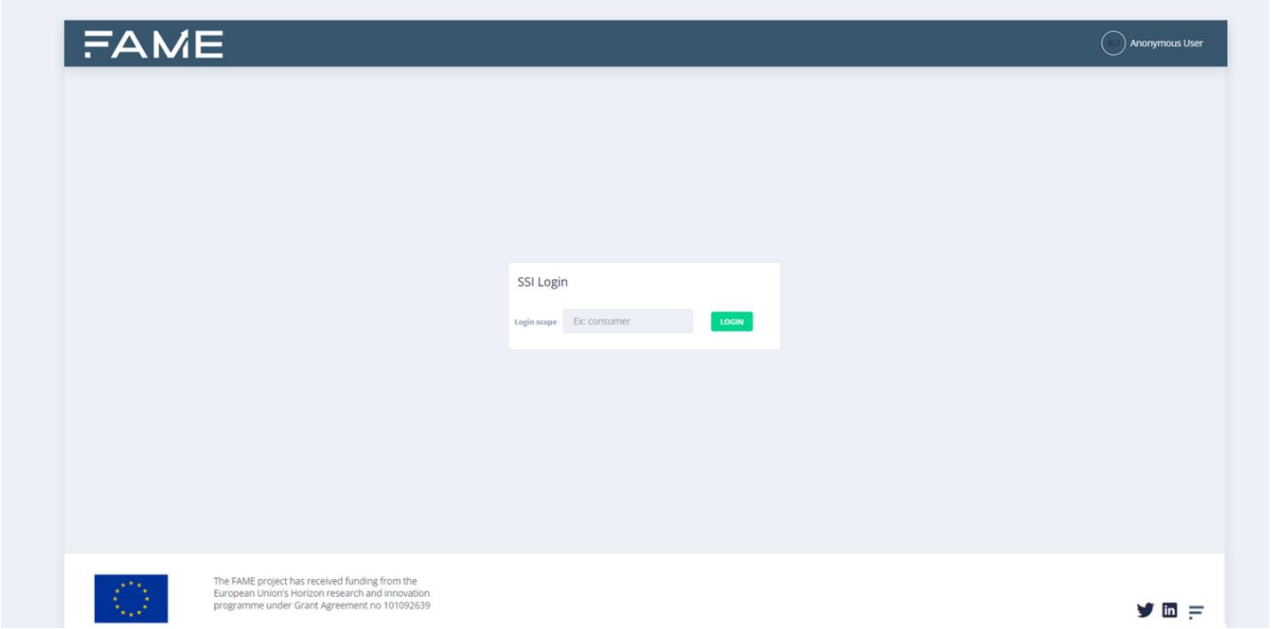

Figure 13 - Demonstrator: login prompt

As we want to demonstrate the Asset Publishing use case, the User sets "provider" as the **Login scope** before clicking on the **LOGIN** button[12]. The Login page then prompts the User to provide a PIN code for *pairing* the server with the User's IDM wallet. The correct PIN is obtained by launching the IDM wallet and by activating the Connect option (Figure 14).

[11] See https://www.w3.org/TR/vc-data-model/

[12] This is actually the name of the identity that was created in the IDM wallet, as explained in §4.1.3. The other valid option is "consumer", but this role is not appropriate for the use case illustrated here.

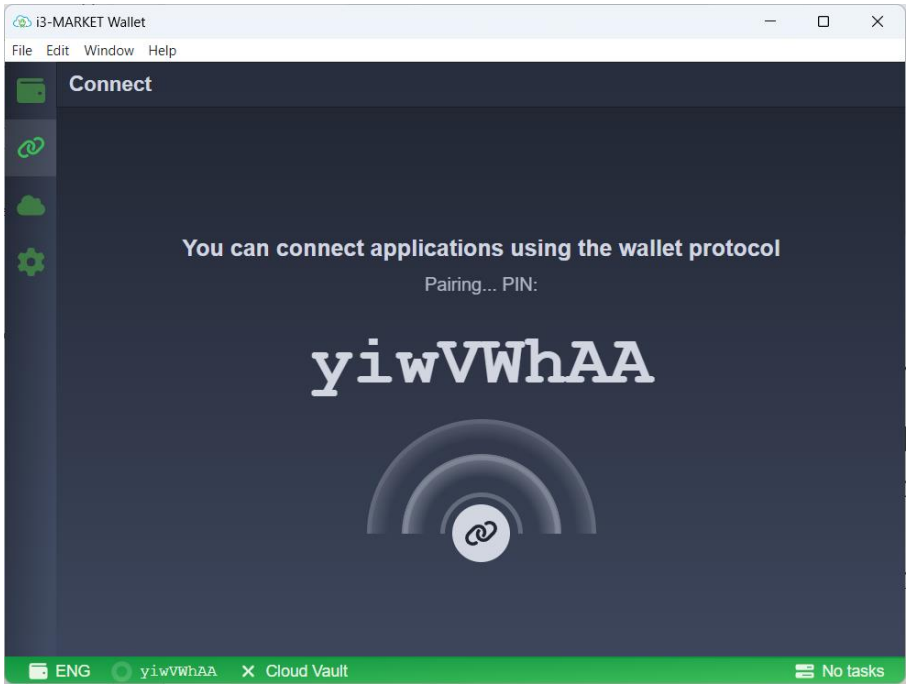

Figure 14 - Demonstrator: pairing the IDM wallet

Once the server and the IDM wallet are paired, the wallet receives a request for the VOC's attributes that are required for logging into the FAME system. The User is notified of the incoming request and reacts by approving the disclosure of such attributes (Figure 15).

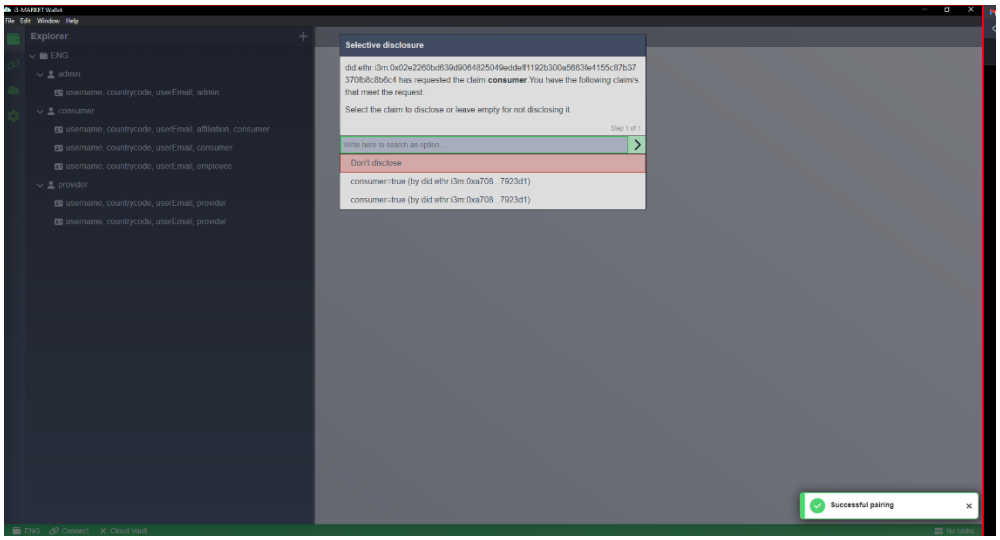

Figure 15 - Demonstrator: disclosure of VOC attributes

Once the VOC is verified by the server, the User is redirected again – this time to the page that was originally requested (Figure 17). Under the hood, this is what actually goes on (Figure 16):

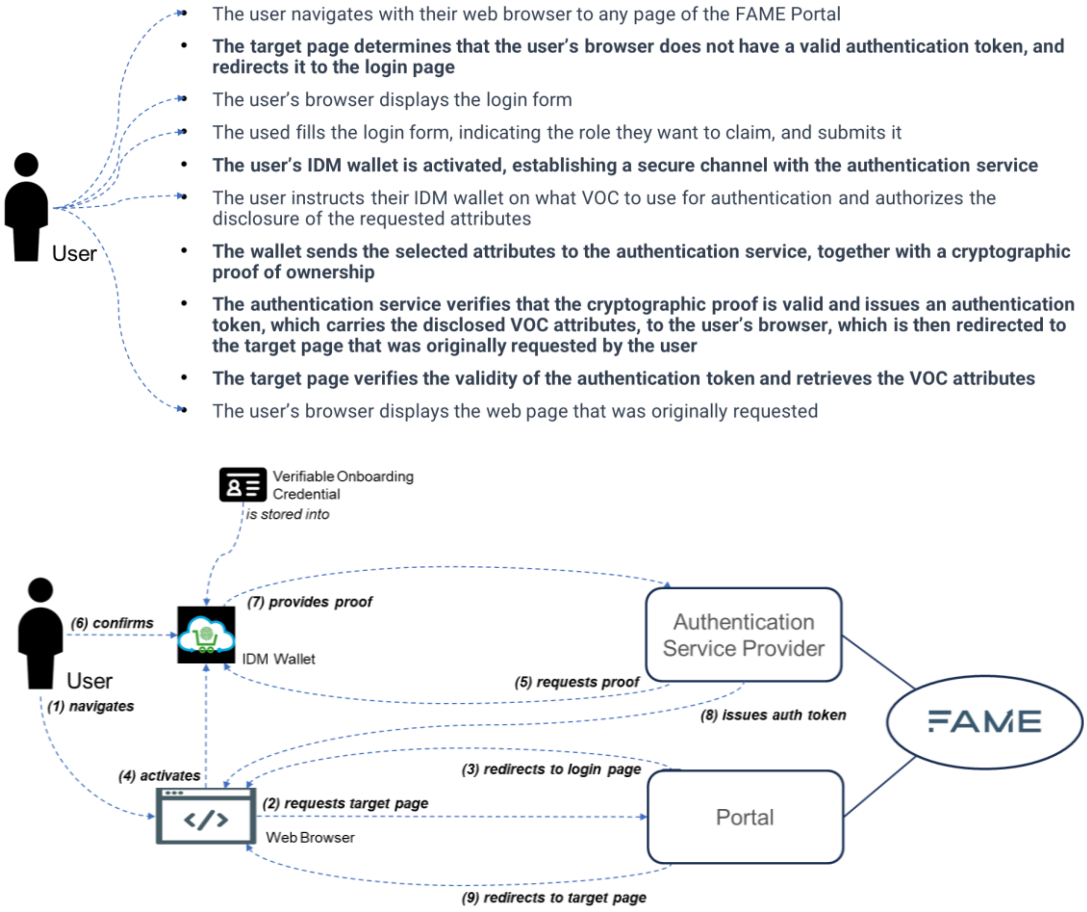

Figure 16 - Demonstrator: integration with AAI

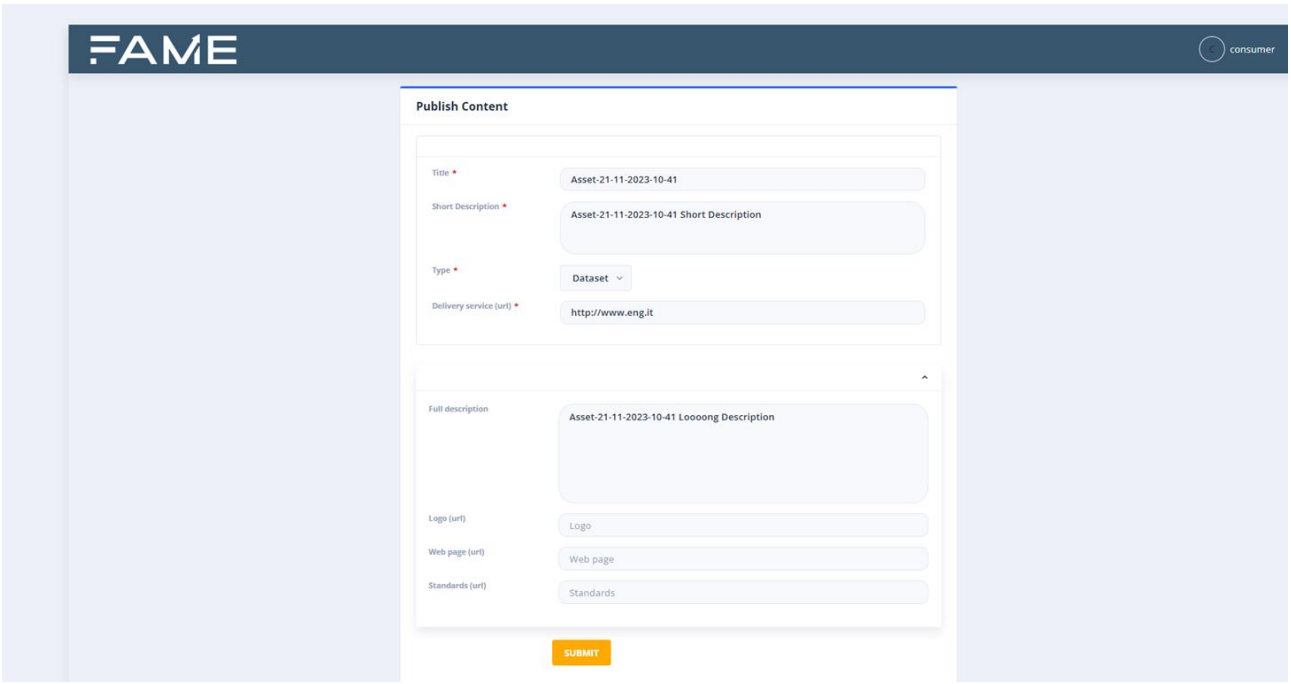

Figure 17 - Demonstrator: asset publishing page

The asset publishing page (Figure 17) collects input from the User: metadata about the asset must be provided and will be published on the FAME federation's catalogue (FDAC). Only the Title, Short description, Type and Delivery service fields are mandatory. Once the User clicks on the SUBMIT button, they receive from the server a blockchain transaction that must be signed by the User and

submitted to the Blockchain Infrastructure in order for the publishing workflow to move forward; this triggers the MetaMask browser extension (i.e., the trading wallet) that presents a pop-up window asking the User to review the transaction before it is signed and submitted (Figure 18).

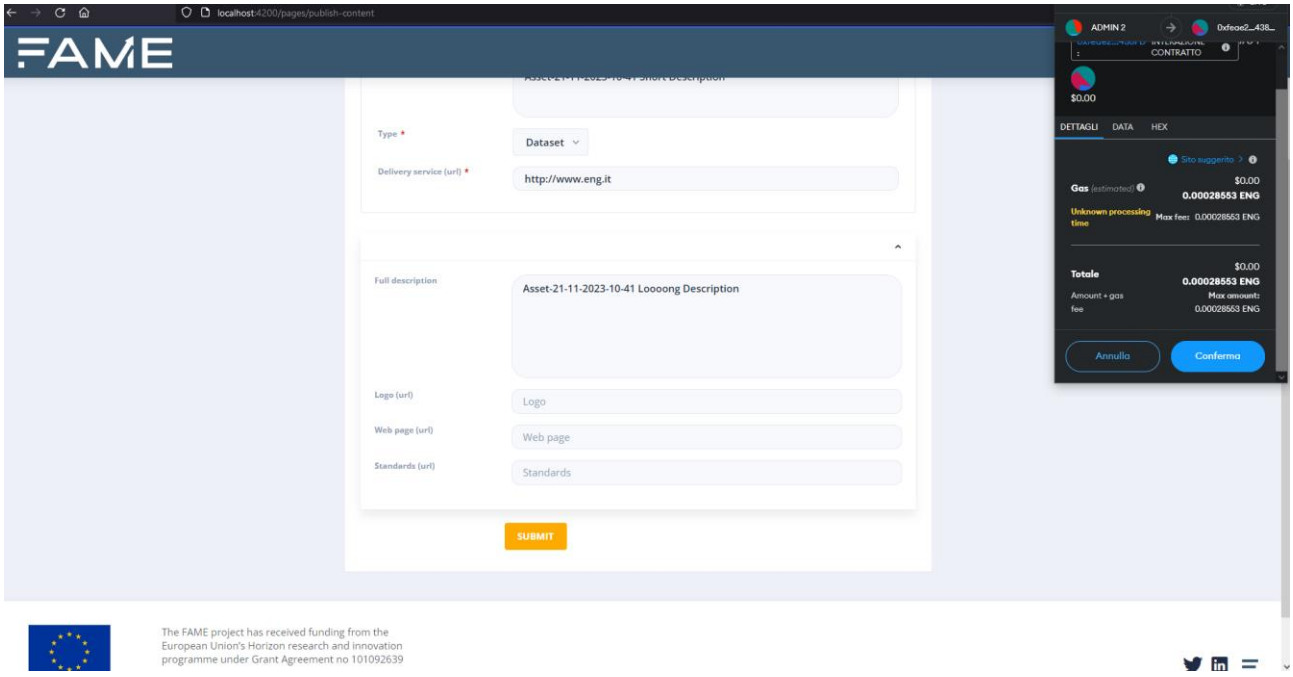

Figure 18 - Demonstrator: signing and submitting a blockchain transaction

When the User confirms, the transaction is processed in the background, according to the workflow explained previously in §2.3/Figure 4. The User can check the progress of the workflow by clicking on the CHECK STATUS button of the asset publishing page (Figure 19). Once the request has PROCESSED status, it means that the asset has been successfully published: a pair of matching entries in the FDAC catalogue and in P&T's Tracing Ledger have been created. The status check window also provides the asset identifier (AID) assigned to these entries.

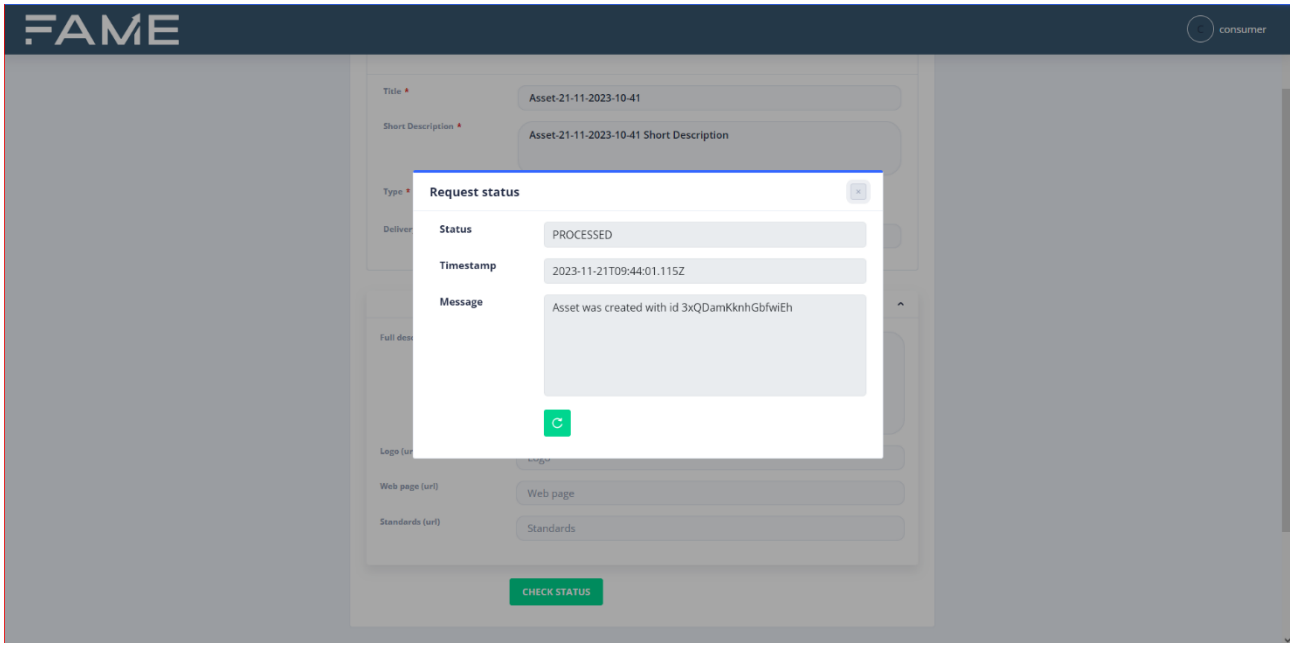

Figure 19 - Demonstrator: checking the status of an asset publishing request

Again, what happens on the FAME platform is even more complex than what the User if aware of, with several modules orchestrated in the background by P&T's Integration Hub (Figure 20).

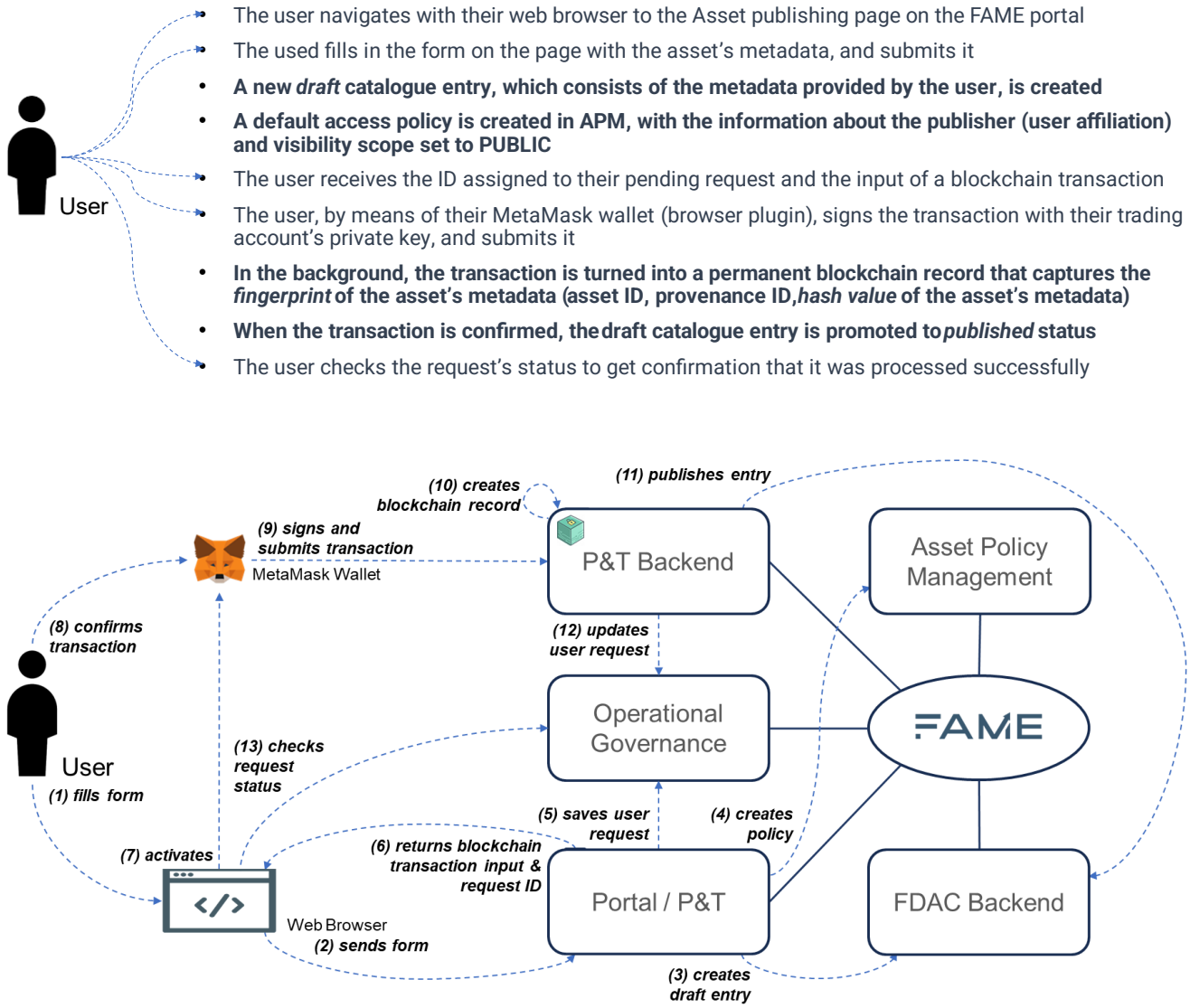

Figure 20 - Demonstrator: integration of P&T with FDAC, APM, and GOV modules

Once an asset is published, commercial offerings can be defined for selling access to its digital contents. This operation is performed by the User – either the same one who originally published the asset, or any other User having the same *affiliation* (i.e., belonging to the same organization, which is registered in the Provenance Ledger as a verified content publisher, as explained in §3.3.1) – through the offering definition page (Figure 21), which can be reached . This form is the entry point to the workflow described in §2.3/Figure 5, and is conceptually similar to the one used for asset publishing, but with two big differences: firstly, it allows the User to interact with the Pricing Advisory Tool (PAT) and receive a suggestion for setting the price tag of the offering; secondly, it does *not* require the User to sign a submit a blockchain transaction.

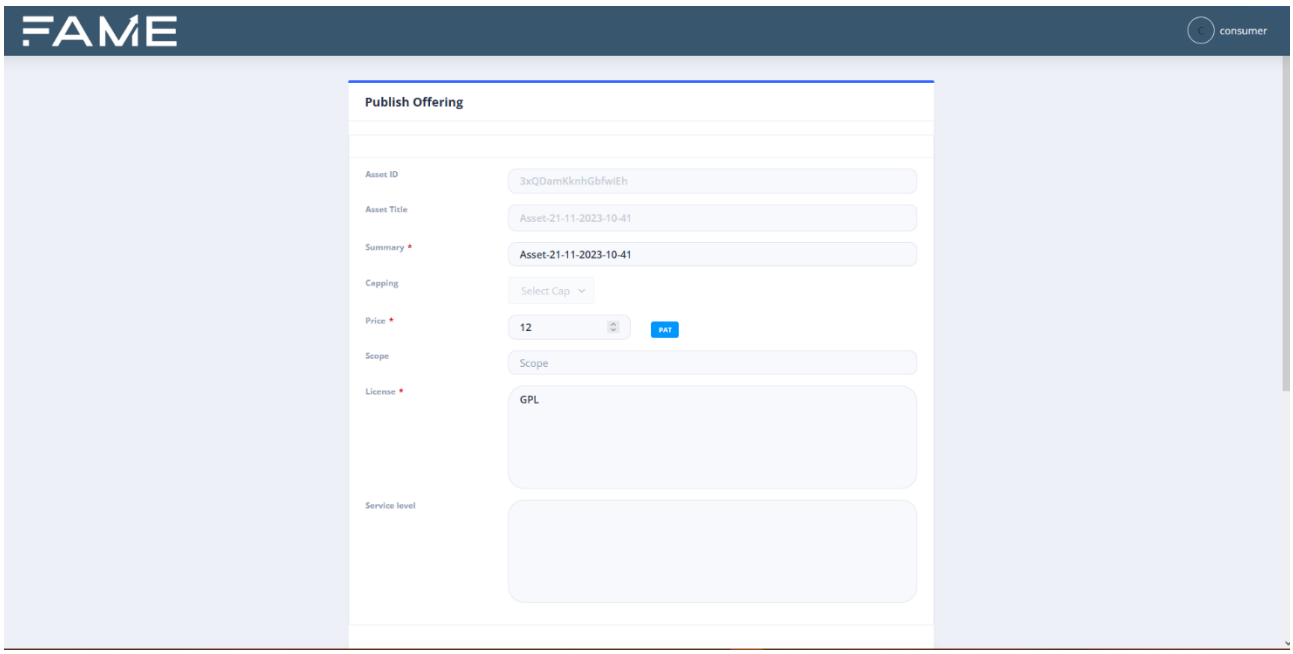

Figure 21 - Demonstrator: offering definition page

The PAT user interface is activated by clicking on the PAT button. This results into a questionnaire being presented to the User in a modal window (Figure 22). We don't go into the details of the PAT process here, as these are out of our scope and will be fully illustrated in a FAME deliverable of future release – D4.2 "Pricing, Trading and Monetization Techniques".

As already seen in the asset publishing use case, the User submits their request for background processing and checks its status, eventually getting the offering identifier (OID) assigned by the system (Figure 23). The full orchestration, with the involvement of the PAT, T&M and GOV modules, is depicted in Figure 24.

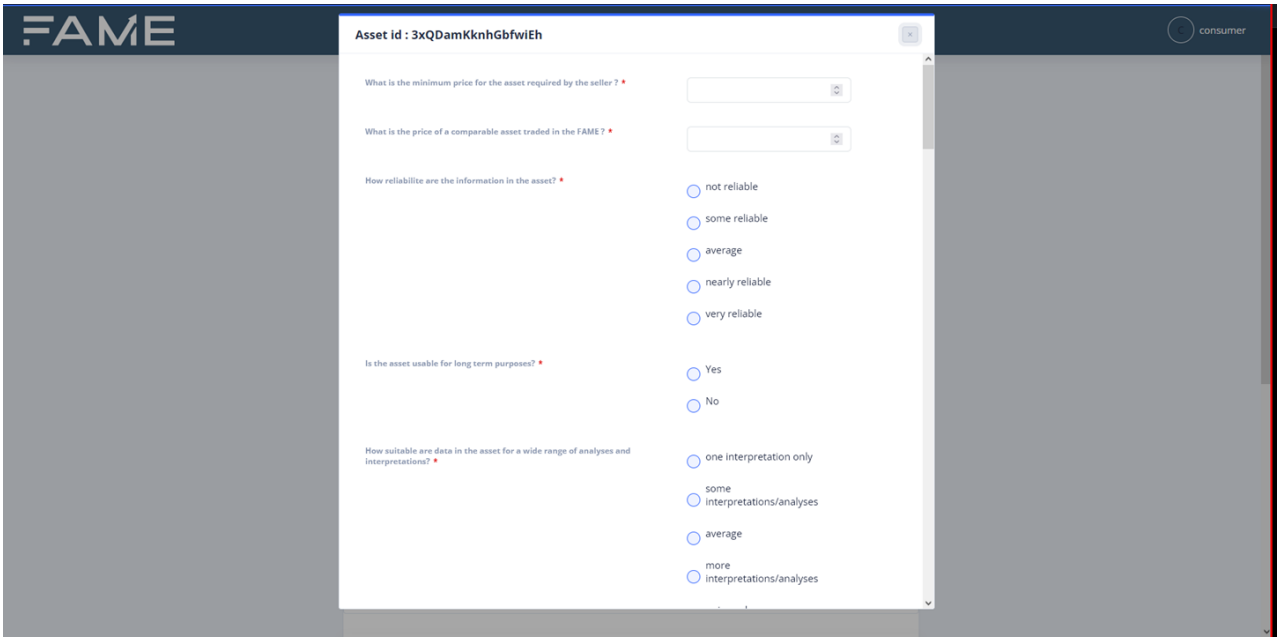

Figure 22 - Demonstrator: PAT in action

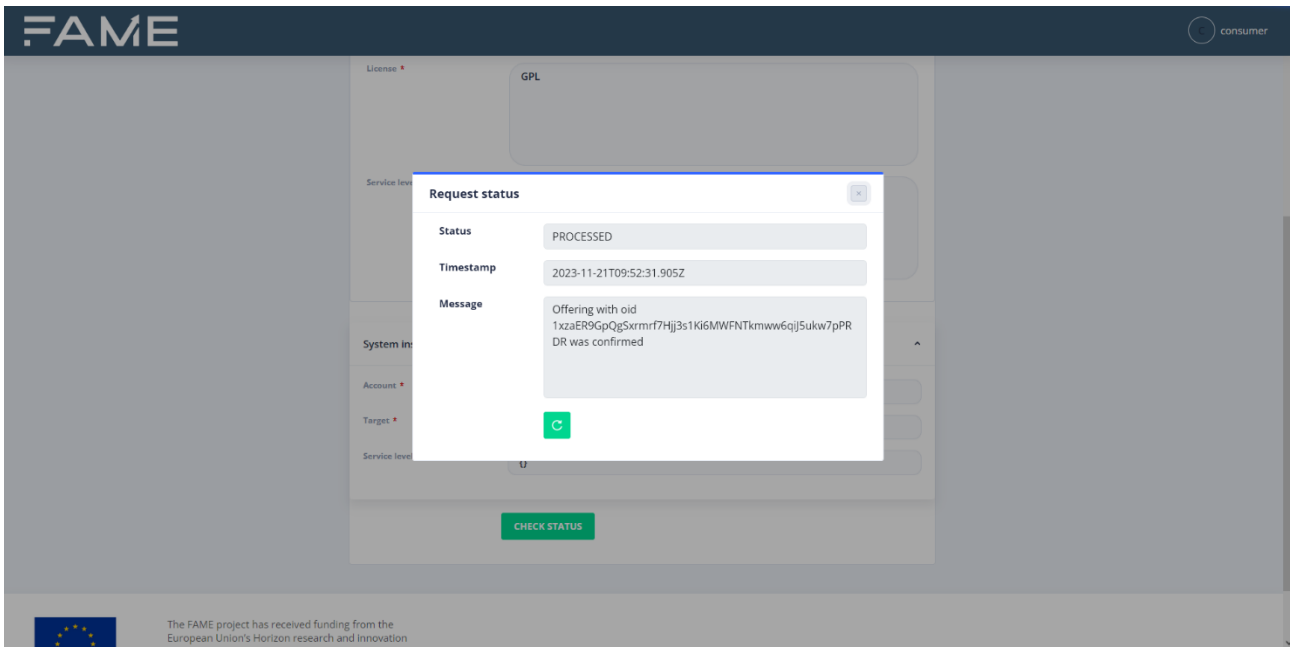

Figure 23 - Demonstrator: checking the status of an offering definition request

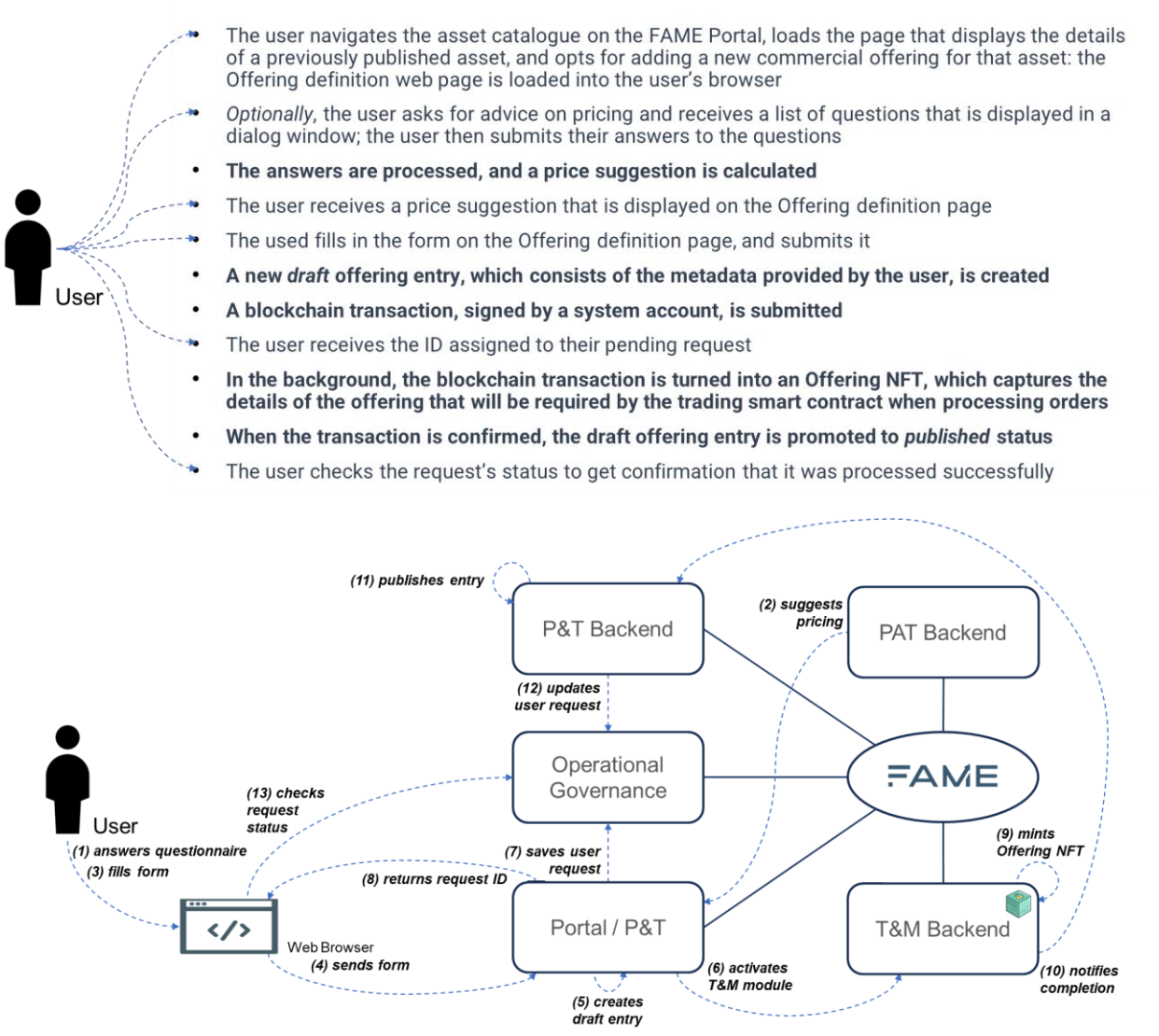

Figure 24 - Demonstrator: integration of P&T with T&M, PAT, and GOV modules

5 Conclusions

In this document, we described the logic and technical specifications of the very first release of the Provenance and Tracing (P&T) module, which implements FAME's Blockchain-based Data Provenance Infrastructure but also plays the role of "orchestrator" for other modules of the FAME platform, in the context of the Asset Publishing and Offering Definition use cases. The demonstrator, which is the result of activities belonging not only to T4.1 "Decentralized Data Provenance and Traceability" but also to several other tasks in WP4 and WP3, was presented – from the end User's perspective – in §4.

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: asset trading in the first place, but also administrative features – like billing, digital currency management, catalogue editing, etc. These capabilities will be mostly enabled by other platform modules, in particular Trading and Monetization (T&M) and Operational Governance (GOV), but P&T will need to be enhanced and extended in order to support them. This 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 belong.