Federated decentralized trusted dAta Marketplace for Embedded finance

D3.1 - Secure Federated Data Management I

Title	D3.1 - Secure Federated Data Management I
Revision Number	3.0
Task reference	T3.1 T3.2
Lead Beneficiary	NUIG
Responsible	Waheed Ashraf, Martin Serrano
Partners	IQB, NRS UBI,
Deliverable Type	DEM
Dissemination Level	PU
Due Date	2023-12-31 [Month 12]
Delivered Date	2024-02-02
Internal Reviewers	ATOS AL
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	2023-11-01	NUIG	Structure and Table of Content
0.2	2023-11-10	NUIG	First Contributions - Technical Specification
0.3	2023-11-20	NUIG	Second Contributions - Technical Specification
0.4	2023-12-01	NUIG	Related Work Update – Table of Content
0.5	2023-12-15	UBI,IQB	Related Work first contribution
0.6	2023-12-20	NUIG	Technical Section Updates
0.7	2023-12-23	NUIG,IQB,UBI	Related Work Update
0.8	2024-01-11	IQB	Latest Version for Internal Review
0.9	2024-01-12	ATOS	Final Corrections after Internal Review
1.0	2024-01-16	GFT,NUIG	Fixing format
2.0	2024-02-01	NUIG	Version for QA
3.0	2024-02-02	NUIG UPRC	Version for submission

Definitions

Acronyms	Definition
AAI	authentication authorization infrastructure
API	Application Programming Interface
APM	Assets Policy Manage
DB	Data Base
DSS	Decision Support Systems|Distributed Software System
EC	European Commission
ERC	Ethereum Request for Comments
FAME	Federated decentralized trusted dAta Marketplace for Embedded finance
FDAC	Federated Data Assets Catalogue
GDPR	General Data Protection Regulation
HTML	Hypertext Markup Language
ID	Identity
JSON	JavaScript Object Notation
JWKS	JSON Web Key Sets
JWT	JSON Web Token
KPI	Key Performance Indicator
OIDC	OpenID Connect
OS	Operating System
REST	Representational State Transfer
RSA	Rivest, Shamir, Adelman Cryptographic Algorithm
SA	Supervisory Authority
SHA	Secure Hash Algorithm
SME	Small and Medium-Sized Enterprises
SSI	Server Side Includes
TR	Technical Requirement
UI	User Interface
URL	Uniform Resource Locator

Executive Summary

In today's digital era, managing multiple usernames and passwords for various online services has become increasingly cumbersome for users. Additionally, organizations are facing challenges in maintaining numerous user accounts while ensuring security and compliance. In this context, federated authentication and authorization infrastructure emerges as a strategic solution to these challenges, streamlining the process of accessing multiple services with a single set of credentials. Complementing this landscape, the Assets Policy Manager (APM) within systems like FAME plays a vital role. It enriches the user experience by integrating with federated identity management, enabling the efficient and secure management of digital assets.

Federated identity management in general serves as a platform offering unified authentication system which enables users to access multiple services across platforms, examples of these federated credential access system is an end-use can access YouTube, Gmail and google docs with a single set of credentials managing thus multiple digital identities. The biggest challenge in a Federated Identity Management systems is while maintaining a simple user experience the credentials to the different systems needs to be maintained secured. There are several attempts to design a centralized identity control within the federated network, but so far even many have been employed to address this challenge and are deployed it still raises privacy and sovereignty concerns.

There has been a shift towards balancing the efficiency of federated systems enabling users to gain more control over their digital identities, thus offering an enhanced user autonomy, privacy, and security. The implementation of centralized services enhances the model portability, but it limits the user control as the ownership and management of the digital identity reside within the centralized system provider, thus making it vulnerable to security threats and leaving the valuable data susceptible to malicious hacking attempts. To address this problem decentralized or self-sovereign identities has emerged as successful solution in the form of Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs). The DID offers a unique global identifier linked to cryptographic keys, enabling secure control and authentication of digital identities. The VCs digitally replicate traditional credentials like driver's licenses or university degrees, embedding essential details such as issuer information and specific attributes. These digital credentials are tamper-evident and provide cryptographically verifiable authorship, revolutionizing how personal qualifications and authorizations are managed and authenticated in the digital domain.

In the other hand the Assets Policy Manager (APM) functionality, which includes the complete lifecycle management of policies for federated assets, aligns with the objectives of federated systems by simplifying how users and organizations manage access to these assets. Through its Rule-Based Access Control (RBAC) model, the APM enhances the security protocols of federated authentication, ensuring that only authorized individuals can access specific assets, thereby upholding both security and compliance standards. In general Assets Policy Manager ensures that other components always display the appropriate assets for authenticated and authorized individuals or organizations. It acts as a central authority, facilitating the enforcement of access controls throughout the system. This synergistic relationship between federated identity management and the APM's asset management capabilities significantly reduces the complexity associated with managing multiple digital identities and assets, offering a more seamless and secure digital experience for both individual users and organizations.

The APM is capable for providing end-users with a comprehensive list of the assets that they have access to their contents. This encompasses assets uploaded by the end-users themselves or any other member of their organization (provenance) which they are affiliated with, as well as assets acquired by them and have active contracts. The APM offers the end-users of a federated system a clear and

consolidated view of their assets, enhancing their ability to manage and track their assets portfolio effectively. This functionality enhances transparency and accountability within the project, allowing users to have a clear view of the assets under their control.

Authorization and access control are imperative for implementation across all facets of an application, including its various access points. Policy Enforcement Points (PEPs) are versatile in their application, suitable for integration into APIs, microservices and Frontend layers, and any other areas within the application where access control is necessary. The widespread incorporation of PEPs throughout the application guarantees frequent and independent verification of authorization at numerous critical points.

This Deliverable reports the specifications and prototype implementations of the FAME Secure Federated Management Framework. The FAME Secure Federated Management Framework is comprised by the FAME Authentication and Authorisation Infrastructure (FAAID) and the FAME Assets Policy Manager (FAPM) System. The following identified functionalities are highlighted and demonstrated as part of the objective(s) for the system prototyping reported in this document:

- Streamline user authentication process across different systems and services while maintaining high security and privacy standards.
- Improve user experience, mitigate security risks, and enhance efficiency in controlling access.
- offer a streamlined, secure, and user-friendly solution for managing digital identities and access controls.
- Enhance the operational efficiency, bolster security, and provide superior user experience, while staying compliant with evolving privacy regulations in an organization.
- Adherence to standards, and continuous adaptation of emerging security threats and technological advancements.
- Enable lifecycle management of policies associated with the federated assets involved in a federated system.
- Facilitate discoverable functionalities through implemented policy management tools, determining what is accessible to be discovered and who is eligible to view and potentially acquire specific assets.
- Provide end-users with a comprehensive list of the assets that they have access to their contents with a clear and consolidated view of their assets, enhancing their ability to manage and track their assets portfolio effectively.

Table of Contents

List of Figures

List of Tables

1 Introduction

In the current digital landscape, user identity information is compartmentalized across multiple centralized databases belonging to various organizations, exposing personal data to an elevated risk of cyber-attacks and breaches. This fragmented infrastructure necessitates the frequent creation of new accounts for different platforms and services, such as online marketplaces, leading to a proliferation of potentially outdated and unsecured personal data repositories.

The paradigm of Distributed or Self-Sovereign Identities (SSI), underpinned by decentralized architectures, presents a transformative resolution to these pervasive challenges. By leveraging blockchain technology and cryptographic principles, SSI places the control of identity data back into the hands of individuals, enabling them to share only what is necessary, with whom they choose, and only for the duration required.

This decentralized identity model significantly mitigates the security risks associated with centralized data storage by eliminating single points of failure. Moreover, it streamlines the user experience by reducing redundant account creation processes, thus simplifying identity verification procedures across a multitude of platforms. Interoperability is inherently designed into these systems, ensuring that identities and credentials can be seamlessly utilized across disparate services and geographic boundaries.

Advancements in this domain are continuously evolving, with research and development focused on enhancing the scalability, user-friendliness, and integration capabilities of SSI solutions. The adoption of SSI represents a significant step towards a more secure, efficient, and user-centric digital identity management framework, realigning the concept of digital identity with the foundational values of privacy, security, and user control.

In the other hand where access control is necessary a suitable solution used is Policy Enforcement Points (PEPs), they are versatile in their application for integration into APIs, microservices and frontend layers, and any other areas within the applications. The widespread incorporation of PEPs throughout the application guarantees frequent and independent verification of authorization at numerous critical points for assets verification and protection. In general Assets Policy Managers (APM’s) ensure that other components always display the appropriate assets for authenticated and authorized individuals or organizations. APM’s act as a central authority, facilitating the enforcement of access controls throughout the system. This synergistic relationship between federated identity management and the APM's asset management capabilities significantly reduces the complexity associated with managing multiple digital identities and assets, offering a more seamless and secure digital experience for both individual users and organizations

The Assets Policy Manager (APM) functionality, which includes the complete lifecycle management of policies for federated assets, aligns with the objectives of federated systems by simplifying how users and organizations manage access to these assets. The APM enhances the security protocols of federated authentication, ensuring that only authorized individuals can access specific assets, thereby upholding both security and compliance standards.

This document describes the specifications and the implementation of the distributed Identity and Access Management system for a FAME federation of data marketplaces called Federated Authentication and Authorisation Infrastructure Deployment (FAAID), providing authentication among all the marketplaces and authorization to access FAME resources and the Federated Assets Policy Manager (FAPM).

1.1 Objective of the Deliverable

The objective of this deliverable is to document both FAME Authentication and Authorization Infrastructure (AAI) and the FAME Assets Policy Manager (APM), including their implementation work towards deploying the prototype that make part of the FAME Marketplace Architecture.

The FAME Authentication and Authorization Infrastructure system's objective is to provide a user-centric authentication based on Self-Sovereign Identity (SSI) and aligning with cutting-edge standardization initiatives such as W3C Decentralised Identifiers (DIDs) v1.0, (the specification is available here: w3c.github.io) [1] and the ongoing work reported in the W3C Verifiable Claims Working Group (available in https://www.w3.org/2017/vc) [2], our approach seeks not only innovation but also widespread commercial acceptance. In striving for comprehensive compliance, we uphold compatibility with widely adopted standards like OpenID Connect (OIDC) while staying abreast of the latest advancements. The objective of the Assets Policy Manager (APM) functionality works towards simplifying how users and organizations manage access to these assets. The APM through its Rule-Based Access Control (RBAC) model included enhances the security protocols of federated authentication, ensuring that only authorized individuals can access specific assets, thereby upholding both security and compliance standards.

The following are the high-level capabilities provided by the Federated AAI and APM subsystems.

Secure Access Control: To ensure that only authenticated and authorized users can access the system or application resources alike the protection of assets.

User Authentication: To verify the identity of users through reliable and secure methods such as SSI, SIOPv2, biometrics, tokens, or multi-factor authentication (MFA).

User Authorization: To determine and enforce what authenticated users are permitted to do within the system or application based on their roles, permissions, or policies.

Compliance: To comply with relevant legal, regulatory, and security standards such as GDPR, HIPAA, or PCI DSS, which often have specific requirements for authentication and authorization.

Auditability: To provide a means for logging and auditing access and actions for security monitoring and compliance verification.

User Experience: To ensure that the authentication process is as frictionless as possible without compromising security, thereby improving user satisfaction.

Scalability and Performance: To build authentication and authorization mechanisms that can scale with the growth of the user and system demands without degrading performance.

Flexibility and Extensibility: To design the system in a way that supports future expansion, such as adding new authentication methods or integrating with other services.

1.2 Insights from other Tasks and Deliverables

The purpose of this deliverable is to document the outcomes of Task 3.1 Federated AAI Infrastructure and Task 3.2 Unified Security Policy Management. This deliverable aims to present the FAME Authentication and Authorization Infrastructure (AAI) along with the Assets Policy Manager (APM). As such, this deliverable receives input and refers to work developed as fully open source in i3-MARKET project. The list of the deliverables identified are i3-MARKET's Deliverable D2.1 (Requirements, Specifications, and Co-Creation), and the i3-MARKET's Deliverable D3.2 (Identity and Access Management Specification and Reference Implementation Report). This deliverable primarily serves as first specification and implementation prototype report and as main input for the subsequent deliverables of FAME Work Package 3 (WP3), specifically for Deliverable D3.4 (Secure Federated Data Management II) where more improvements/functionalities can be reported.

1.3 Structure

The deliverable is structured as follows:

- **Chapter 1 Introduction**: This section serves as the gateway to the deliverable, detailing the main objectives and goals of the document within the context of the FAME project.
- **Chapter 2 Positioning into FAME SA**: In this section, the deliverable is contextualized within the larger framework of the FAME Solution Architecture (SA).
- **Chapter 3 Components Specification**: This crucial section delves into the specifics of key components within the FAME project. It includes a comprehensive examination of the SSI Micro Services, starting with a description that outlines their purposes and roles, followed by a discussion of related work and literature. The technical specifications are detailed, along with its interfaces and how they interact with other components. Similarly, the Assets Policy Manager (APM) is explored, beginning with a description, then moving into relevant related work, its technical specifications, and its interfaces within the project. This section helps to understand functionalities of the core components of the FAME project.
- **Chapter 4 Components Demonstration**: This section is dedicated to demonstrating the practical application of the key components discussed earlier. It includes detailed information on the VC and OIDC Service, starting with prerequisites and the installation environment, followed by a step-by-step installation guide and a user guide for end-users. Similarly, the Assets Policy Manager (APM) is covered, providing necessary pre-installation information, an installation guide, and a user manual. This section is instrumental in illustrating how the components are implemented and used within the FAME project.
- **Chapter 5 Conclusions**: The final section of the deliverable encapsulates the key findings, insights, and outcomes derived from the analysis and demonstrations of the FAME SA and its components. It provides a summary of the deliverable, touching on the significant achievements and the Key Performance Indicators (KPIs) met, as outlined in the document.

2 Positioning into FAME SA

In the C4 architecture model of the FAME Solution Architecture [3][4], the Federated Authentication and Authorization Infrastructure is classified as a container named "Authentication and Authorization". This container is included in the "Federation Manager" system. Its role is to provide authentication and authorization using distributed identity and verifiable credentials. To achieve this, it receives asset metadata from the Federated Data Asset Catalogue and supports the "Transaction Operations" system by providing these asset metadata. Additionally, it manages the authorization credentials received from the OpenAPIs.

Furthermore, the Asset Policy Manager is classified as a container named "Assets Policy Management", also included in the "Federation Manager" system. Its primary role is to ensure the secure management of assets within FAME. It interacts with the container named "Regulatory Compliance" to update information regarding applicable regulations. It also exchanges asset policy rules with the data assets catalogue and runs a function for FAME-compliant policy formats.

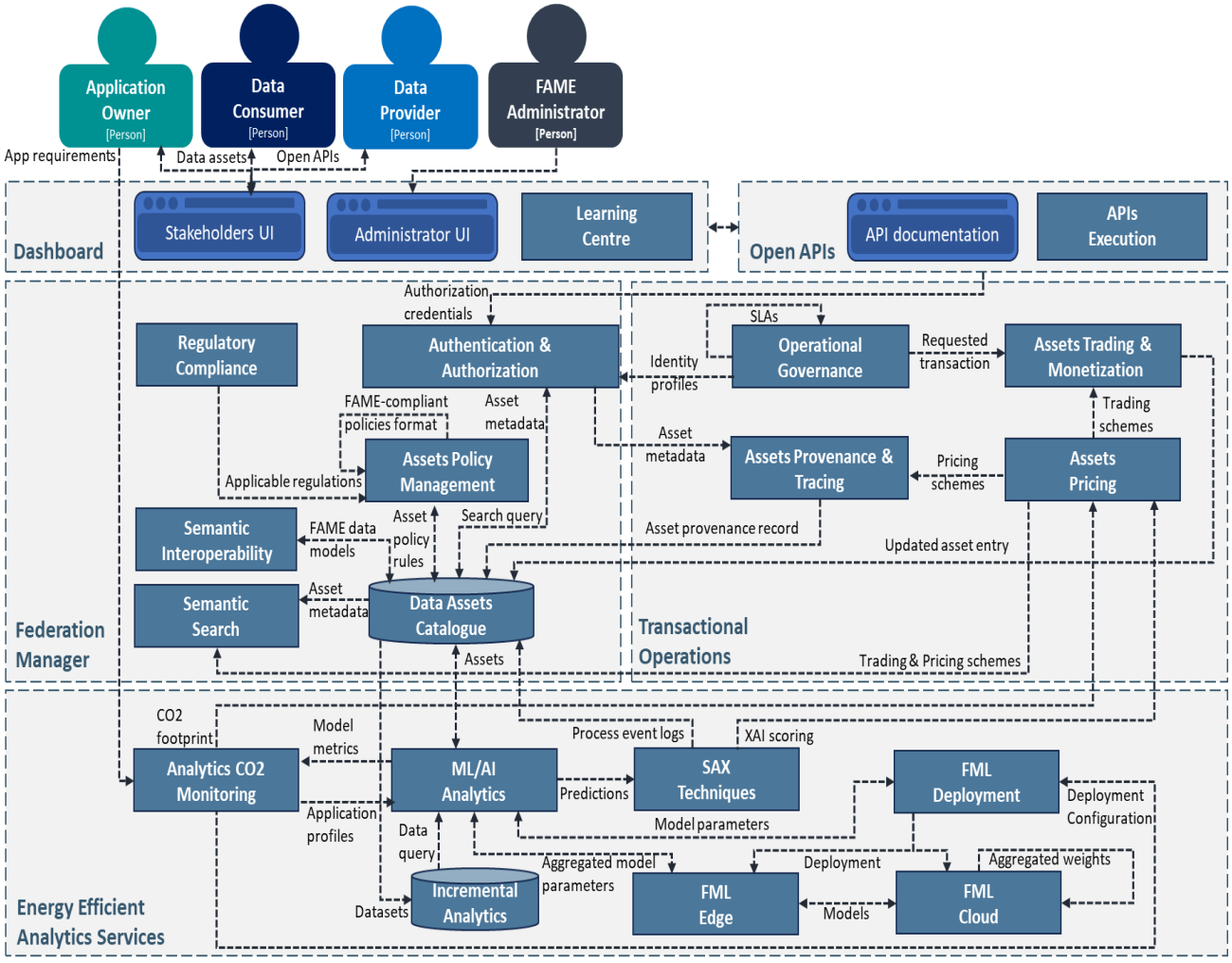

Figure 1 – FAME SA C4 Container Diagram [4]

3 Components Specification

3.1 Authentication & Authorization Infrastructure (AAI)

3.1.1 Description

Leveraging the FAME issuer API, we aim to implement a state-of-the-art system for issuing Verifiable Credentials compliant with the W3C Verifiable Credentials Data Model [5]. This advanced issuance system will be underpinned by the OID4VCI protocol [6], which itself is a sophisticated extension of the OpenID [7] framework specifically designed to augment the interoperability and security of digital identity verification processes.

The system will facilitate a robust, cryptographically secure mechanism for the issuance and management of Verifiable Credentials. By incorporating OID4VCI, we ensure adherence to a standardized communication protocol that promotes trust and integrity in the exchange of identity information. This protocol enables a secure, consent-based sharing of credentials, reinforcing user control over personal data while streamlining the verification process.

The issuance workflow will be encapsulated in an API-driven architecture, ensuring seamless integration and a high degree of automation. The API will support various credential formats and encapsulate complex operations such as proof generation and signing, simplifying the integration for relying parties.

Additionally, the system will feature comprehensive compliance with the latest security protocols and privacy standards, providing a robust defence against common vulnerabilities and threats in the digital identity landscape. The goal is to deliver an issuance platform that not only meets the current demands for digital verification but also is agile enough to adapt to future advancements in the field of decentralized identity.

3.1.2 Related Work

The introduction of OpenID Connect for Verifiable Credentials (OIDC4VC), a new technology standard published on November 26, 2023, [1][8], marks a significant advancement in the realm of digital identity verification, particularly from the perspective of credential issuers and the verifier component. This new standard addresses the growing need for more secure and efficient methods of issuing digital credentials, reflecting the latest developments in the field.

Building upon existing foundations, the FAME Authentication & Authorization Infrastructure utilizes the i3-MARKET SSI technology [9], rather than starting from scratch. FAME's AAI microservices, including OIDC SSI Auth and Verifiable Credential microservice, are integrated with the Veramo framework. This integration is pivotal for managing Decentralized Identifiers (DIDs) and verifiable credentials effectively. The infrastructure's implementation will support pilot scenarios, and some features will be evaluated in specific cases.

The list of related technical requirements related to Federated Authentication and Authorization Infrastructure as these have been documented in deliverable D2.1 (Requirements Analysis, Specifications and Co-Creation) is quoted below:

- TR002: be able to access the assets of several marketplaces and data spaces using a single sign-on mechanism.
- TR022: Be able to register myself and/or my organization.
- TR901: Support interfaces for data assets trading, pricing, and data policy management.
- GR_002: Support federated identity management.

3.1.2.1 Advancements beyond the Related Work

The integration of the OIDC4VC technology into the FAME project represents a significant leap in digital identity management and data exchange. This innovative approach, focuses around the issuer and verifier microservices of OIDC4VC, aligns seamlessly with the self-sovereign identity model that FAME embraces, enhancing user control and privacy. By incorporating OIDC4VC's issuer microservice, FAME leverages the power of Distributed Identifiers (DID) and Verifiable Credentials (VC) to create a more secure and user-centric system.

This integration allows users to generate and manage their digital identities autonomously, ensuring unique identification within the federation. Such identification is essential for maintaining data integrity and confidentiality during exchanges. The OIDC4VC framework bolsters trust in these transactions by guaranteeing the security and verifiability of issued credentials.

A key benefit of OIDC4VC in the FAME environment is the simplification of the credential verification process. Credentials issued via OIDC4VC can be easily authenticated by third parties without direct involvement from the issuer. This efficiency is crucial for FAME, aiming to minimize data integration challenges among stakeholders. It streamlines the verification process, ensuring it doesn't become a bottleneck in the system's operation. Furthermore, the OIDC4VC issuer microservice embodies FAME's principles of data minimization and explicit user consent. It enables the sharing of only necessary data during transactions, with users having full control over their information. This approach not only respects user privacy but also maintains the accuracy and integrity of federation data.

The addition of the OIDC4VC verifier microservice to the FAME project marks a critical development in digital identity and data exchange. This microservice, functioning as an OAuth 2.0 Client, adds an extra layer of security and trust, crucial in environments where data exchange integrity is paramount. The verifier microservice transcends traditional verification methods, offering a more secure and user-focused approach. It authenticates credentials securely, upholding data accuracy, integrity, and confidentiality. This verifier microservice addresses the challenges of traditional identity models by simplifying user authentication across the federation without compromising security and privacy. Its flexible design boosts interoperability within the diverse stakeholder network of FAME, enabling seamless interaction with various wallet applications and user devices.

In terms of compliance, the OIDC4VC verifier aligns with privacy-by-design principles, mirroring FAME's commitment to user consent and data minimization. This alignment is crucial in the ever-changing landscape of data protection regulations and privacy laws. Moreover, it resonates with FAME's adoption of the self-sovereign identity model, emphasizing user empowerment and control over personal data and credentials. This enhancement of the user-centric approach fosters trust and improves user experience within the federation.

OIDC4VC's integration into the FAME project signifies a paradigm shift in digital identity verification and data exchange. It strengthens the project's foundation in user control, privacy, and security, aligning with modern regulatory requirements and user expectations in the digital age. This integration is not just a technological advancement but a step towards a more empowered and secure digital ecosystem.

3.1.3 Technical Specification

3.1.3.1 Component-level C4 Architecture

To provide authentication and authorization with distributed identity and verifiable credentials, we implemented two Node.js micro services. The Issuer Verifiable Credential micro service provides the APIs that implement the core functions to manage verifiable credentials, namely issuing, verifying, and revoking verifiable credentials, and a utility function.

In the management of verifiable credentials, the Verifiable Credentials microservice employs the Veramo framework for credential generation. Subsequently, it interacts with the FAME Wallet API to seamlessly deliver the generated credential to the user. In the Verifiable Credential issuance during user registration, the microservice undergoes a two-step process. Initially, it authenticates the Decentralized Identifier (DID) of the user. Subsequently, it issues a verifiable credential for the authenticated DID, certifying the user's role—either as a data consumer, data provider, or both.

The OIDC SSI Auth micro service provides the API to perform the authorization code flow with PKCE using verifiable credentials as proof method.

The distinctive feature of the OpenID Connect provider, as opposed to traditional providers, lies in its requirement for the disclosure of verifiable credentials. The specification of credentials to be disclosed is achieved using the scope field. The OpenID Connect provider offers static scopes such as "openid" (mandatory for the standard), "profile" (including user profile info in the id token), and "vc" (adding verifiable claims into the id token). In comparison to the standard OpenID Connect scope, the additional scopes are "vc" and "vce."

The first step to use the OIDC provider is to register a new client in the provider. There is an easy procedure to do this. Using the GET /developers/login API with you receive an initial access token. Later, this initial access token has to be used as bearer token in the API POST /oidc/reg. This API creates a new client. The client parameters are specified in the body. Well, now that you've created your client it's time to authenticate and authorize! Through the OIDC provider it is possible to implement the authorization code flow + PKCE flow.

An important aspect of configuring a proper authentication flow is to correctly specify the scope parameter. It must contain the following claims: - openid: Mandatory for the Open ID Connect standard. It returns the sub field of the id token, and its value is the user did. - vc: It adds the field vero able claims into the id token. Useful when the Open ID Connect Provider wants to check any information about the vero able claims asked. Compared to the standard scope of Open ID Connect, the scopes added are vc and vce. On the other hand, the standard email scope, which returns the user's email, is not present. There are two different types of scopes: - vc:vc_name: It asks the user for the specific verifiable claim vc name. It is optional, so the user can decide whether to append it or not. If the issuer of the verifiable claim is not trusted, it will be added into untrusted verifiable claims array of the id token. - vce:vc_name: It asks the user for the essential verifiable claim with name: vc name. It is mandatory, so if the user does not provide it or the issuer is untrusted, the login and consent process will be cancelled.

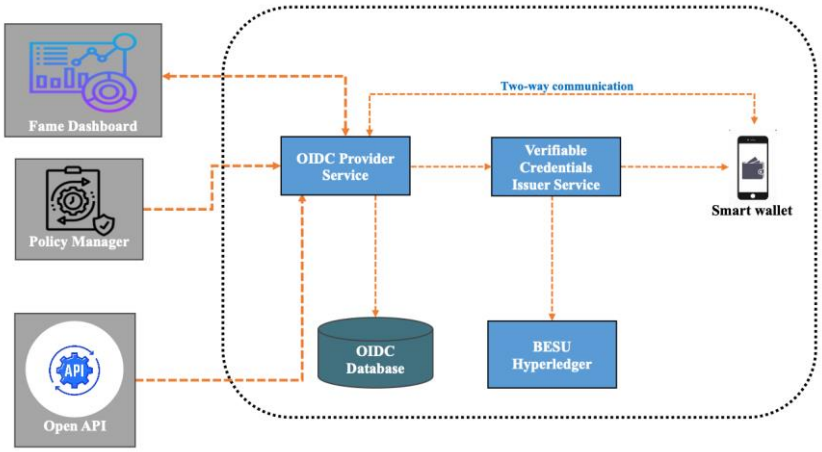

Figure 2 – AAI C4 Container Diagram

3.1.3.2 Baseline Technologies and Tools

As part of the implementation and to address open-source solutions, we have selected Veramo [10] as the framework of choice, a This decision comes as a strategic upgrade from the previously used uPort library [11], which is now not supported by the open-source community. Veramo stands out with its advanced features and robust support, positioning it as the optimal choice for our current requirements and future scalability.

Integrating the Veramo framework into both the OIDC SSI Auth and the Verifiable Credential microservice, we leverage its comprehensive feature set for efficient management of Decentralized Identifiers (DIDs) and verifiable credentials. This integration enhances the capabilities of these components, utilizing the robust functionality of Veramo for optimized digital identity and credential handling.

The FAME Network is a consortium of various Data Marketplaces, each operating its instance of the FAME service. These marketplaces are equipped with their dedicated OIDC SSI Auth Service and a specialized Verifiable Credential microservice, responsible for the creation, verification, and revocation of verifiable credentials. In alignment with the W3C Recommendations on verifiable credentials, the OIDC SSI Auth Service functions as the verifier, while the Verifiable Credential microservice acts as the issuer, endowed with additional capabilities. The end-user is the holder, possessing their verifiable credentials.

Each Verifiable Credential microservice instance maintains its unique Decentralized Identifier (DID) and private key, utilized for signing the verifiable credentials. As a result, every verifiable credential issued is distinctly linked to the DID of the microservice that generated it. Furthermore, in terms of revocation, exclusive authority is vested in the originating microservice, ensuring that only it can revoke the credential it issued. This structure not only reinforces the integrity and traceability of the credentials but also ensures a secure and streamlined revocation process.

The user securely stores their Verifiable Credentials in their digital wallet and provides explicit consent to share these credentials with the OIDC SSI Auth Service as needed during the authentication process. This approach ensures that the user retains full control over their personal credentials while enabling seamless and secure verification when accessing services.

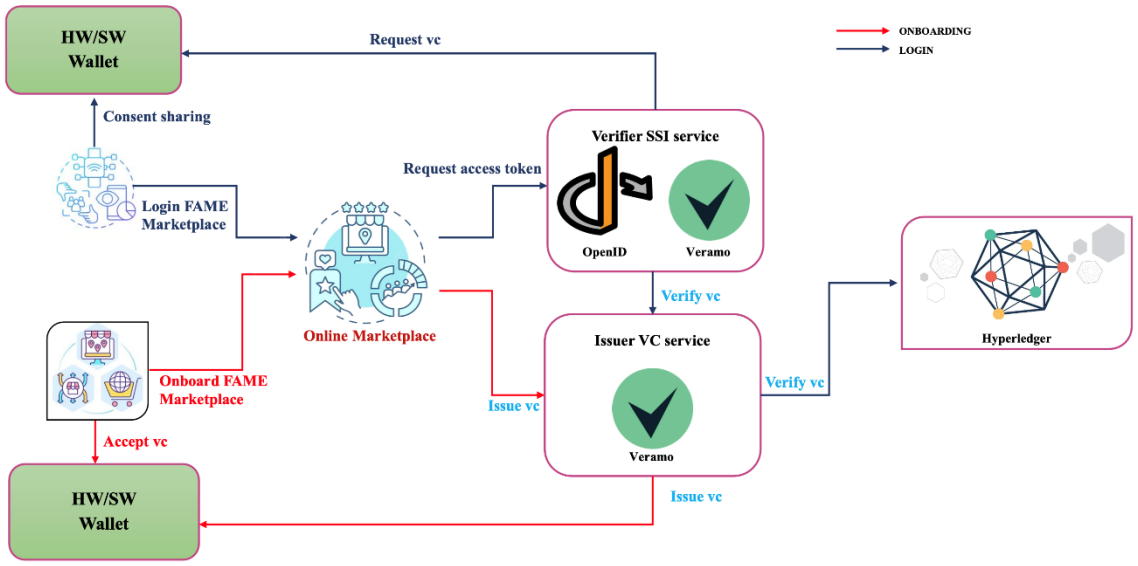

Figure 3 – AAI Solution Design

The subsequent subsections provide a comprehensive presentation of modules and detailed workflows.

3.1.3.3 DID management

The Management of Decentralized Identifiers (DIDs) is facilitated through the Hyperledger Besu blockchain, utilizing the Veramo ethr-did library for robust and secure DID operations.

The library adheres to the ERC-1056 standard, designed for leveraging Ethereum addresses as fully autonomous Decentralized Identifiers (DIDs).

The Ethr-DID offers a scalable identity solution for Ethereum addresses, endowing them with the capability to aggregate both on-chain and off-chain data effectively.

This DID method operates in conjunction with the Ethr-DID-Registry, a smart contract designed to streamline public key resolution for both off-chain and on-chain authentication. The Ethr-DID-Registry not only facilitates key rotation but also manages delegate assignment and revocation, enabling third-party signers to act on behalf of a key. Additionally, it supports the setting and revocation of off-chain attribute data. The cumulative effect of these interactions and events is leveraged by the Ethr-DID-Resolver to construct the complete DID document.

```
{
  '@context': 'https://w3id.org/did/v1',
  id: 'did:ethr:0xb9c5714089478a327f09197987f16f9e5d936e8a',
  publicKey: [{
       id: 'did:ethr:0xb9c5714089478a327f09197987f16f9e5d936e8a#owner',
       type: 'Secp256k1VerificationKey2018',
       owner: 'did:ethr:0xb9c5714089478a327f09197987f16f9e5d936e8a',
       ethereumAddress: '0xb9c5714089478a327f09197987f16f9e5d936e8a'}],
  authentication: [{
       type: 'Secp256k1SignatureAuthentication2018',
       publicKey:
'did:ethr:0xb9c5714089478a327f09197987f16f9e5d936e8a#owner'}]
}
```

```
{
  "id": "did:ethr:0x02051f2c2193fd4453dfd56d826c0c46bcb8df4f40a6bdc5e5ca48c866f86657b5",
  "verificationMethod": [
    {
      "id": "did:ethr:0x02051f2c2193fd4453dfd56d826c0c46bcb8df4f40a6bdc5e5ca48c866f86657b5#controller",
      "type": "EcdsaSecp256k1RecoveryMethod2020",
      "controller": "did:ethr:0x02051f2c2193fd4453dfd56d826c0c46bcb8df4f40a6bdc5e5ca48c866f86657b5",
      "blockchainAccountId": "eip155:1:0xe6ED3da06d791702b2cB4bF98531DC0548783061"
    },
    {
      "id": "did:ethr:0x02051f2c2193fd4453dfd56d826c0c46bcb8df4f40a6bdc5e5ca48c866f86657b5#controllerKey",
      "type": "EcdsaSecp256k1VerificationKey2019",
      "controller": "did:ethr:0x02051f2c2193fd4453dfd56d826c0c46bcb8df4f40a6bdc5e5ca48c866f86657b5",
      "publicKeyHex": "02051f2c2193fd4453dfd56d826c0c46bcb8df4f40a6bdc5e5ca48c866f86657b5"
    }
  ],
  "authentication": [
    "did:ethr:0x02051f2c2193fd4453dfd56d826c0c46bcb8df4f40a6bdc5e5ca48c866f86657b5#controller",
    "did:ethr:0x02051f2c2193fd4453dfd56d826c0c46bcb8df4f40a6bdc5e5ca48c866f86657b5#controllerKey"
  ],
  "assertionMethod": [
    "did:ethr:0x02051f2c2193fd4453dfd56d826c0c46bcb8df4f40a6bdc5e5ca48c866f86657b5#controller",
    "did:ethr:0x02051f2c2193fd4453dfd56d826c0c46bcb8df4f40a6bdc5e5ca48c866f86657b5#controllerKey"
  ],
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/secp256k1recovery-2020/v2",
    "https://w3id.org/security/v3-unstable"
  ]
}
```


This library has been seamlessly integrated into both the OIDC SSI Auth and Verifiable Credentials microservices to facilitate the resolution and authentication of DIDs, enabling direct interaction with the user's wallet. It is compatible with the Decentralized Identifiers specification proposed by the W3C Credentials Community Group.

The library is utilized by the Verifiable Credentials microservice for the creation and management of distributed identities that issue credentials, whereas the Distributed Identities pertaining to users are generated and maintained within the users' own wallets.

3.1.3.4 Verifiable Credential management

The Verifiable Credentials microservice harnesses the Veramo framework for the generation of credentials and interfaces with the FAME Wallet API to deliver the credentials directly to the user, streamlining the management of verifiable credentials.

3.1.3.5 Issue a Verifiable Credentials

In the innovative registration process at a Data Marketplace, the user is issued a Verifiable Credential, marking the commencement of their engagement. Here, a specialized microservice first authenticates the user's Decentralized Identifier (DID). Upon successful authentication, it issues a Verifiable Credential linked to this DID, certifying the user's designated role within the marketplace, which could be as a data consumer, data provider, or a combination of both.

The interactions and multiple activities involved in this process are delineated in the following diagram, which is the generalisation of the user registration process as specified in the i3-MARKET project as documented in the i3-MARKET D3.2 (Identity and Access Management Specification and Reference Implementation Report). The purpose of including this generalisation diagram is to clearly outline the roles and interactions of the various entities involved in the user registration.

1. **Identity Holder (FAME user):** The individual or entity associated with a specific identity within the FAME system, representing a user who holds verifiable credentials and interacts with the platform.

2. **User Agent (Client of OIDC - FAME Data Marketplace website):** The software or application acting on behalf of the FAME user to communicate with the OpenID Connect (OIDC) protocol, facilitating secure authentication and authorization on the FAME Data Marketplace website.

3. **FAME Wallet**: A secure digital storage facility where verifiable credentials are stored for the FAME user. It serves as a repository for authentication and authorization-related information within the FAME ecosystem.

4. **Verifiable Credential Microservice (FAME Data Marketplace instance):** A specialized service within the FAME Data Marketplace that handles the generation, verification, and management of verifiable credentials. It plays a crucial role in ensuring the integrity and security of identity-related information.

5. **FAME Data Marketplace Backend**: The backend infrastructure of the FAME Data Marketplace, responsible for handling the core functionality, data processing, and communication between different components. It forms the backbone of the entire system, facilitating seamless interactions and data exchange.

Initially, the user inputs their registration details (1), which categorize them as either a data consumer or a data provider. Once a user is registered under one of these roles, they gain access across all sites within the Fame Network. Consequently, the FAME Data Marketplace entity exclusively issues two distinct verifiable credentials for these roles. This information is provided by users through registration forms located in a specific section of the FAME Data Marketplace.

To ascertain the appropriate DID (Decentralized Identifier) for issuing the verifiable credential, the FAME Data Marketplace needs to acquire the user's DID. This process is facilitated through the wallet-protocol session API, adeptly crafted for establishing a secure connection ("pairing") with the user's wallet. It involves the use of a one-time password (OTP) to securely retrieve the user's DID. Specifically, the FAME Data Marketplace executes a GET request to the issue API of the Verifiable Credentials microservice (2). This request includes parameters such as a callbackUrl, which specifies the URL to redirect the user post-credential issuance, and the credential itself, formatted as a JSON-encoded object.

The FAME Data Marketplace initiates this API call, then the Verifiable Credentials microservice employs the "pairing" protocol to establish a connection with the user's wallet. During this process, it requests a One-Time Password (OTP) to facilitate a secure connection with the FAME Smart Wallet (3). This step is crucial for ensuring a secure and authenticated link between the marketplace and the user's digital wallet. The user creates a new One-Time Password (OTP) by utilizing the corresponding function in their wallet. This OTP is then provided to the Verifiable Credentials microservice, initiating a secure session (4). This process is an essential security measure, ensuring that the session is both authenticated and encrypted for safe data exchange.

Subsequently, the Verifiable Credentials microservice dispatches a share request to acquire the user's Decentralized Identifier (DID) (5). This request is a critical step in the process, enabling the secure retrieval of the user's unique identifier necessary for proceeding with the transaction.

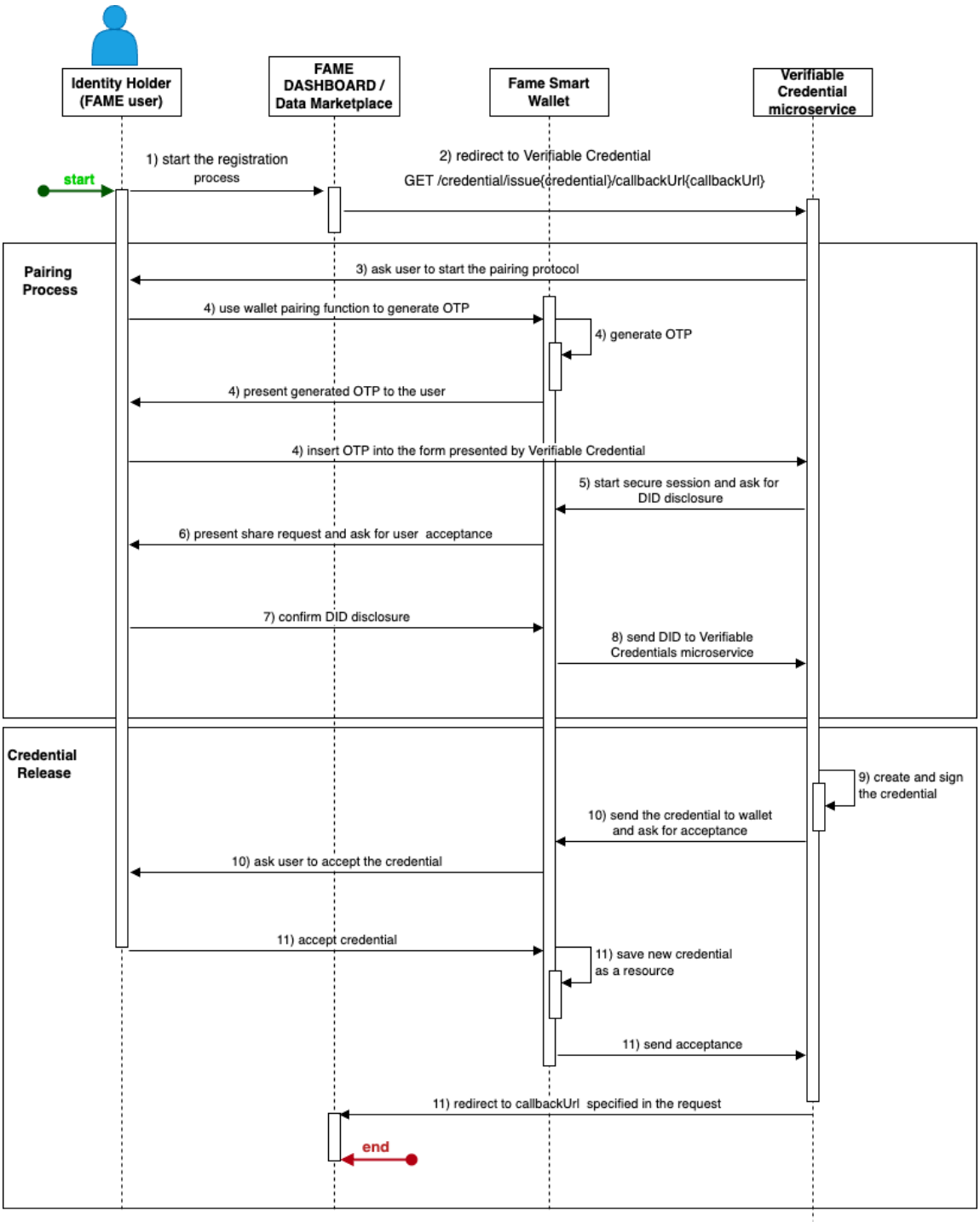

Figure 4 – User registration process [12]

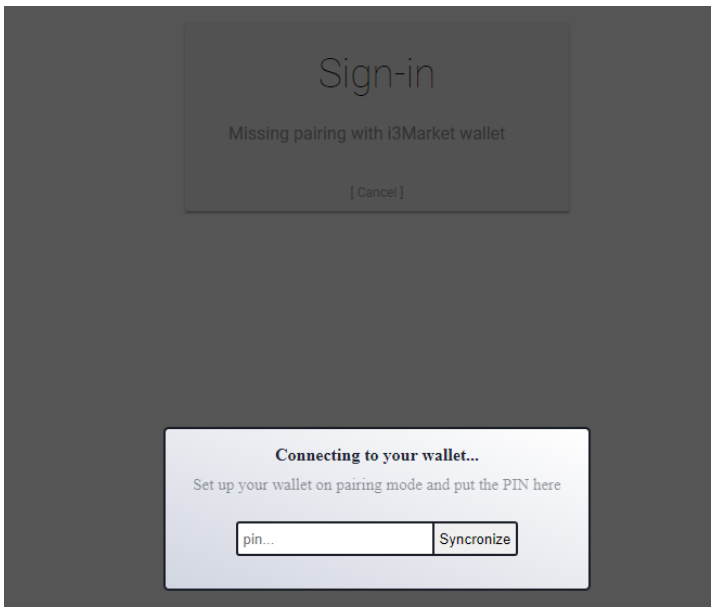

Figure 5 – OTP request to start pairing process.

At this juncture, the user receives a disclosure request via their wallet, prompting them to make a decision: whether to accept or decline the sharing of their requested identity, the Decentralized Identifier (DID) (6). This step places control in the hands of the user, ensuring they have the autonomy to consent to or refuse the sharing of their personal identifier.

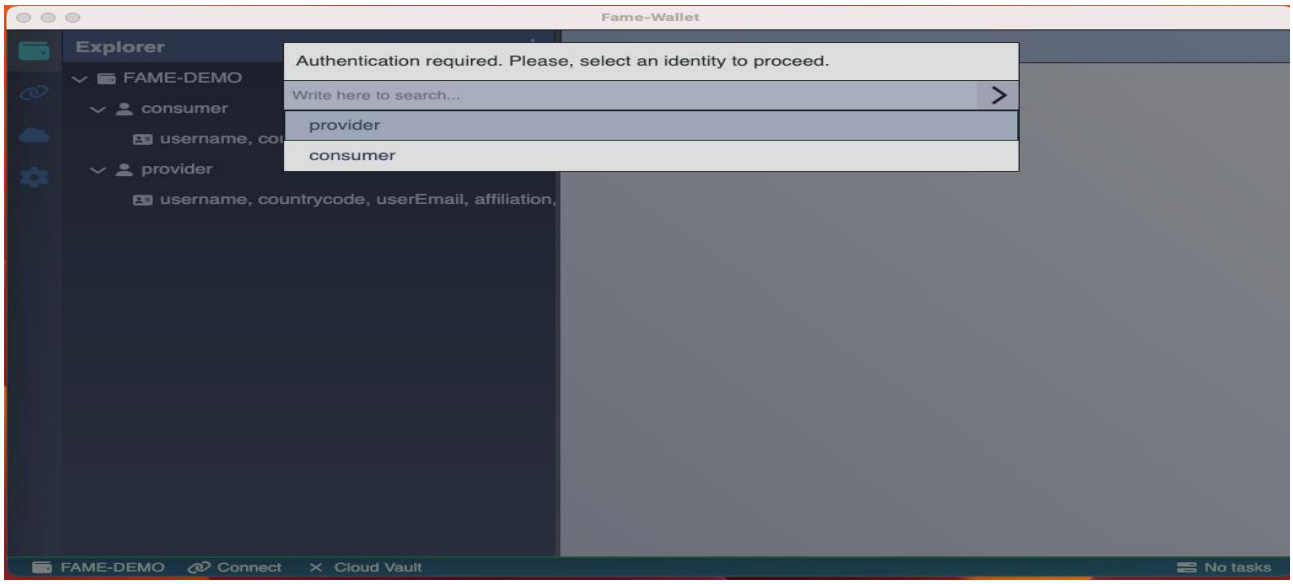

Figure 6 – FAME smart wallet request to disclose the DID.

Should the user consent to share their DID (7), the FAME Smart Wallet proceeds to transmit the access token to the Verifiable Credentials microservice (8) using a callback mechanism. Following this, the Verifiable Credentials service decodes the access token to extract the DID of the authenticated user. This step is crucial for ensuring that the user's identity is securely and accurately communicated to the service.

The second phase of the process focuses on issuing a verifiable credential to affirm the user's status as a data consumer. Broadly, this procedure encompasses two primary steps:

- Employing cryptographic signatures to secure the credential data.
- Transmit the signed credential as a JSON Web Token (JWT) to the FAME Smart Wallet.

To generate a verifiable credential, the Verifiable Credentials microservice executes an internal API call to the POST /credential/issue/{DID} endpoint (9). This call transmits the recently retrieved user's DID and the credential details, packaged as form-data in the specified format:

```
{
" data_consumer": true
}
```

The FAME Data Marketplace is limited to requesting the issuance of credentials solely pertaining to the registration process, specifically for data consumers and data providers. In contrast, credentials related to the acquisition of assets or services can be requested by Data Providers. Upon the API's invocation, the Verifiable Credentials microservice executes the 'create Verifiable Credential' function from the Veramo Core library's DID Agent, which is a key component of the Veramo framework.

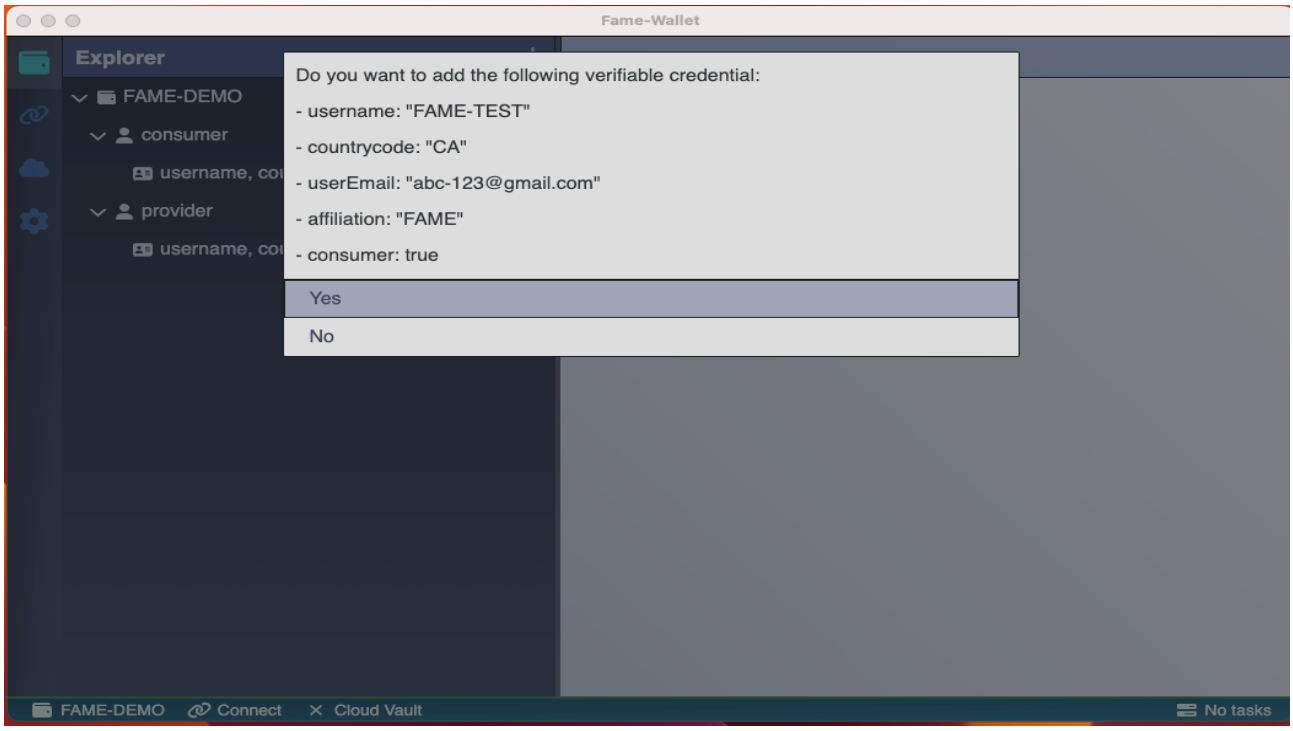

Figure 7 – Verifiable Credential Acceptance.

When the Fame Smart Wallet receives a credential, it conducts a verification process for its signature. Each signed message is accompanied by an “iss” attribute, which includes the Decentralized Identifier (DID) of the issuer. To authenticate the public key associated with the message, a DID resolver is utilized.

The Veramo DID Agent is compatible with an array of DID methods. These include “did:ethr” (based on ERC-1056), offering blockchain-based identity solutions; “did:web,” which, when used in tandem with blockchain-based DIDs, leverages the existing reputation of a web domain to establish trust; and “did:key,” a self-certifying DID method that operates independently of external systems like the Blockchain, enhancing its efficiency and reliability. More details about supported DID methods can be found in Veramo documentation online here:

https://veramo.io/docs/veramo_agent/did_methods

Given that Hyperledger Besu is the reference blockchain in this context, the users' Decentralized Identifier (DID) adopts the format “did:ethr”.

Following the completion of signature verification, the wallet prompts the user to approve the credential (10). Once the user accepts the credential (11), it is securely stored in their wallet. Subsequently, it becomes accessible and visible within the 'Resources' tab, which displays a comprehensive list of all credentials registered in the app.

At this point the user will be redirect to the *callbackUrl*, previously specified.

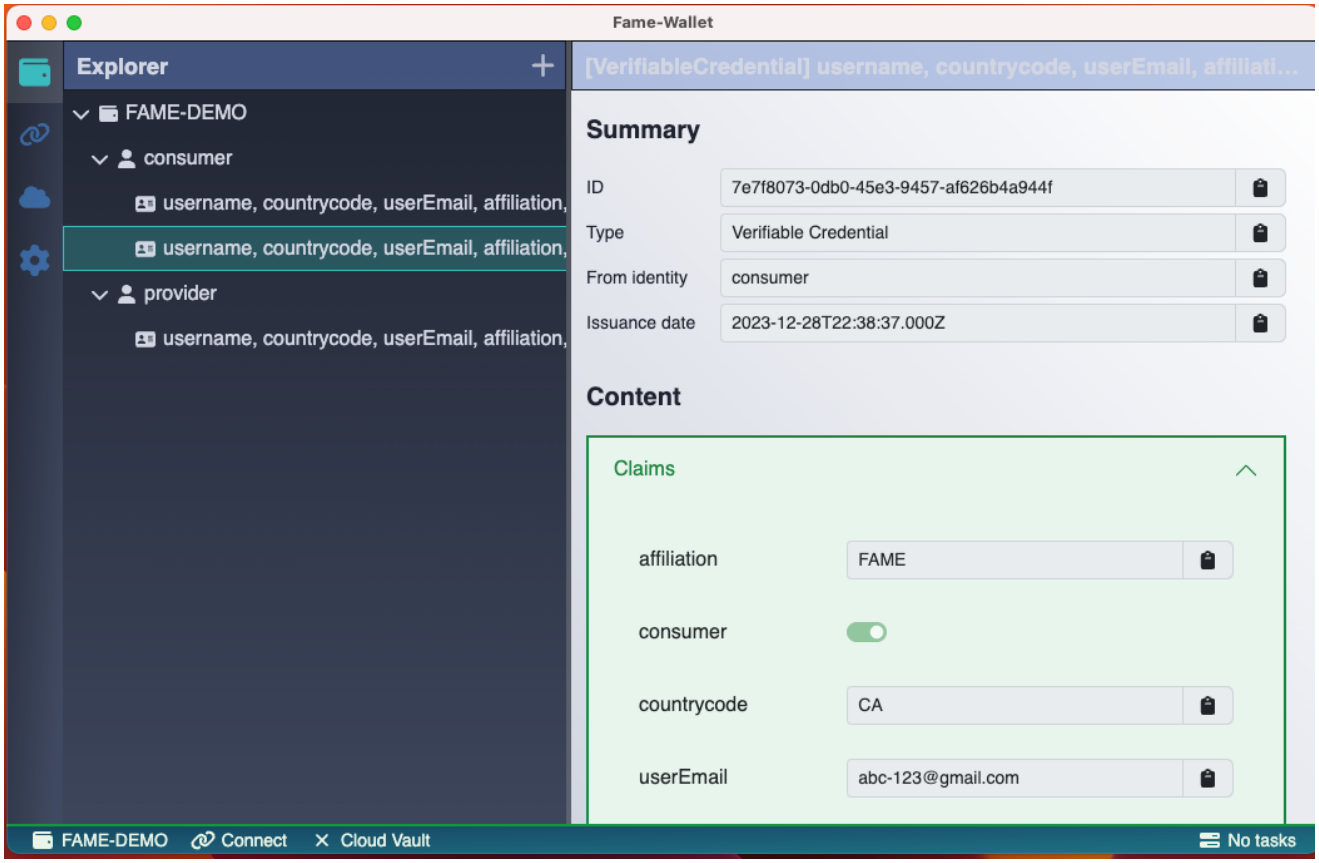

Figure 8 – Credentials List.

After securely storing credentials in their digital wallet, a user can confidently present them during authentication requests as proof of possessing the necessary credentials to access specific resources or services.

3.1.3.6 Revoke a Verifiable Credential

In the realm of verifiable credentials, the issuance of credentials is a fundamental aspect. However, equally important is the capability to revoke them when necessary. This revocation function is a pivotal element within a verifiable credential ecosystem, ensuring credentials remain current and valid. Take, for instance, a scenario where a FAME data provider issues a credential for service access. If a data consumer breaches the terms of use, the data provider must have the means to respond effectively. In such instances, the provider may decide to suspend the consumer's service access due to the violation. This necessitates altering the status of the verifiable credential, so that when a relying party subsequently verifies the status, it reflects the consumer's invalid status and lack of authorization for service access. To facilitate this essential process, an API for credential revocation has been

developed. The workflow for credential revocation is illustrated in the accompanying figure, providing a clear and structured approach to managing credential statuses.

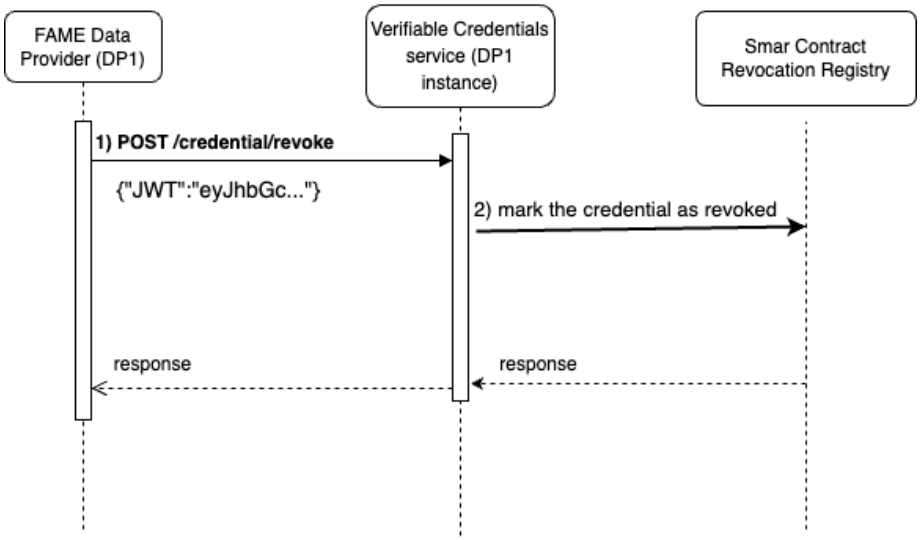

Figure 9 – Revoke Verifiable Credential Flow.

At the beginning of the flow, the data provider calls the API of the verifiable credentials micro service (1) communicating that a specific credential belonging to a user must be marked as revoked. The verifiable credential to be revoked is passed through the body parameter in the form {“JWT”: “eyJhbGc ..."}.

As an implementation choice, it was decided that only those who issued a credential are allowed to revoke it. To satisfy this requirement, a check is made, if the issuer of the credential is the address of the issuer, then it proceeds, otherwise it immediately blocks the flow. When the verifiable credentials micro service receives the API call, it writes the credential hash through a transaction in a smart contract named Revocation Registry (2), to keep track of the action performed. Since the FAME blockchain is an Ethereum-type blockchain, the smart contract is written in solidity and its code is the following:

```
pragma solidity ^0.5.8;

contract RevocationRegistry {

    mapping(bytes32 => mapping(address => uint)) private revocations;

    function revoke(bytes32 digest) public {
        require (revocations[digest][msg.sender] == 0);
        revocations[digest][msg.sender] = block.number;
        emit Revoked(msg.sender, digest);
    }

    function revoked(address issuer, bytes32 digest) public view returns (uint) {
        return revocations[digest][issuer];
    }

    event Revoked(address issuer, bytes32 digest);
}
```

The smart contract Revocation Registry provides two functions:

- A publicly accessible function for revoking a credential.
- A publicly accessible function to verify whether a credential is included in the revocation list, indicating its revocation status.

The revoke function requires a 32-character string as input and records this association with the transaction sender, signified by the committed address in the line. To ensure a consistent 32-character length, the credential undergoes processing through a SHA-3 cryptographic hash algorithm before being marked on the smart contract, and the resulting 32-character digest is then written.

The smart contract's data structure comprises a private array named "revocations," which stores digest-address associations (line 5). When a credential is added to the register, it is linked through the credential digest and the issuer of the transaction, specifically the message sender. This mapping includes the block number, effectively the transaction counter ID.

As a deliberate implementation choice, the decision was made to restrict the revocation of a verifiable credential to the service that originally issued it. This measure is in place to prevent unauthorized third parties from revoking credentials they did not issue. It is inherently logical that only the provider granting access to the service should have the authority to revoke it.

As evident from the smart contract code, a prerequisite for adding a verifiable credential to the smart contract is ensuring it is not already recorded in the register (line 8). In other words, the corresponding mapping should not contain a block number indicating the transaction that added the credential. If the credential is not present, it can be written to the contract (line 9). Upon successful addition, an event is triggered (line 11), signalling the issuer of the transaction and the digest of the newly added credential in the register (line 17).

One potential issue in the current smart contract implementation is its implicit trust in the integrity of the data being written to the registry, specifically that the data is a valid digest of a JWT-formatted credential. As it stands, the smart contract lacks an access control mechanism to restrict writing privileges to specific addresses. Consequently, once deployed on the blockchain, any valid address can invoke the public methods of the smart contract, potentially leading to the recording of inconsistent or erroneous data in the registry. A viable solution to mitigate this risk is the establishment of a roster of trusted issuers for transactions. An issuer can be deemed trustworthy if it adheres to the requisite cryptographic hash algorithms for the verifiable credential prior to its registry entry. Plans to incorporate these critical security measures will be undertaken in subsequent phases of the project's development.

Table 1 – Baseline Technologies and Tools

Baseline Technology	Description	Added value to FAME
i3-MARKET SSI	Identity framework	i3-MARKET already performed a research work and implemented SSI, This technology helping to establish a decentralized authentication and authorization solution for the FAME marketplace.

3.1.4 Interfaces

This section describes the classes and endpoints of the defined interfaces. Verifiable Credentials (VC) service acts as an issuer role in the self-sovereign identity model. The tables included contains the details of the end point descriptions and Components URI alike the API body examples.

Table 2 – Issue Credentials.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#FAAI-1
Endpoint Name	Issue a credentials
Endpoint Description	This APIs produces an HTML page that must be displayed on a browser. It is therefore necessary to redirect the user at the url of this API. The generated HTML page contains a script that communicates with the FAME wallet
Component	VC
Endpoint URI	/issue/{credential}/callbackUrl/{callbackUrl}
HTTP Method	GET

Request (Query) Parameters	{
	
	Consumer: true
	
	}

Request Body (example)	N/A
Request Headers	N/A
Response Body	Produce HTML page, then rendering of this page, a notification will appear on the wallet asking you to select the identity (DID) on which to save the verifiable credential. After selecting it, it automatically generates the credential and sends it back to the FAME wallet. The wallet will present a new notification asking if you want to accept the credential. Upon acceptance, the credential will be saved and viewable in the wallet.
Response Status Code	200 - OK

Table 3 – Issue a Revoke Credentials.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#FAAI-2
Endpoint Name	Revoke credential
Endpoint Description	This APIs produces an HTML page that must be displayed on a browser. It is therefore necessary to redirect the user at the url of this API. The generated HTML page contains a script that communicates with the FAME wallet
Component	VC
Endpoint URI	/credential/revoke
HTTP Method	POST
Request (Query) Parameters	N/A

Request Body (example)	{
	
	"credentialJwt": "string"
	
	}

Request Headers	N/A
Response Body	Produce HTML page, then rendering of this page, a notification will appear on the wallet asking you to select the identity (DID) on which to save the verifiable credential. After selecting it, it automatically generates the credential and sends it back to the FAME wallet. The wallet will present a new notification asking if you want to accept the credential. Upon acceptance, the credential will be saved and viewable in the wallet.
Response Status Code	200 - OK

Table 4 – Verify a Credential.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#FAAI-3
Endpoint Name	Verify credential
Endpoint Description	This APIs produces an HTML page that must be displayed on a browser.
Component	VC
Endpoint URI	/credential/verify
HTTP Method	POST
Request (Query) Parameters	N/A

Request Body (example)	{
	
	"credentialJwt": "string",
	
	"credentialIssuer": "string"
	
	}

Request Headers	N/A
Response Body	Produce HTML page, then rendering of this page, a notification will appear on the wallet asking you to select the identity (DID) on which to save the verifiable credential. After selecting it, it automatically generates the credential and sends it back to the FAME wallet. The wallet will present a new notification asking if you want to accept the credential. Upon acceptance, the credential will be saved and viewable in the wallet.
Response Status Code	200 - OK

Table 5 – Login to get token.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#FAAI-4
Endpoint Name	Login to get token
Endpoint Description	The first step to use the OIDC provider is to login and get a JWT token to register a new client in the provider.
Component	OIDC node
Endpoint URI	/developers/login
HTTP Method	GET
Request (Query) Parameters	Username & password
Request Body (example)	N/A
Request Headers	N/A

Response Body	{
	
	"initialAccessToken": "JmmRhudSc6VkVzvIQamP-pe5Zj4k7TR_oqe6uzIbiYW"
	
	}

Response Status Code	200 - OK

Table 6 – Client Registrations.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#FAAI-5
Endpoint Name	Client Registration
Endpoint Description	The first step to use the OIDC provider is to register a new client in the provider using JWT token.
Component	OIDC node
Endpoint URI	/oidc/reg
HTTP Method	GET
Request (Query) Parameters	N/A

Request Body (example)	{
	"grant_types": [
	"authorization_code"
	],
	"token_endpoint_auth_method": "client_secret_jwt",
	"redirect_uris": [
	"http://localhost:3000/api/credential"
	],
	"post_logout_redirect_uris": [
	"http://localhost:3000/auth"
	],
	"client_name": "fame_demo_app_05",
	"id_token_signed_response_alg": "EdDSA"
	}

Request Headers	N/A
Response Body	{ "application_type": "web", "grant_types": ["authorization_code"], "id_token_signed_response_alg": "EdDSA", "post_logout_redirect_uris": ["http://localhost:3000/auth"], "require_auth_time": false, "response_types": ["code"], "subject_type": "public", "token_endpoint_auth_method": "client_secret_jwt", "introspection_endpoint_auth_method": "client_secret_jwt", "revocation_endpoint_auth_method": "client_secret_jwt", "require_signed_request_object": false, "request_uris": [], "client_id_issued_at": 1703799657, "client_id": "HHn7wzoY_v_zWd-zht3yj", "client_name": "fame_demo_app_05", "client_secret_expires_at": 0, "client_secret": "FXrl6dePLjdVUNp7qNg0UWThBHBEOvsvi7OGV6qONd9CZmpueYssOm-KpayQeiDxXBStqeCG4TJdTc4WoW6ohg", "redirect_uris": ["http://localhost:3000/api/credential"], "registration_client_uri": "https://identity.fame-horizon.eu/release2/oidc/reg/HHn7wzoY_v_zWd-zht3yj", "registration_access_token": "w860m6GOI_4S5GV-18bNfZsAqLvHv5IQem1n-r0mXla" }
Response Status Code	201 - Created

Table 7 – Authentication and Authorization.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#FAAI-6
Endpoint Name	Authentication and Authorization
Endpoint Description	The authorization code flow involves calling two different APIs
Component	OIDC node
Endpoint URI	/oidc/auth /oidc/token
HTTP Method	GET, POST
Request (Query) Parameters	Scope, response_type, client_id, redirect_uri, state, code_challenge

Request Body (example)	N/A
Request Headers	N/A
Response Body	As you can see, this first API returns an HTML page as a response. This page is the login page to be rendered by the user's browser. This means that the user must be redirected to this page via a redirect. Under the hood, this page contains scripts that allow pairing with the wallet. In particular, if a valid pairing is not detected, an automatic procedure is activated. A form is then shown in which the user must copy a code generated through the UI of the i3-Market desktop wallet. Once the pairing code has been entered correctly, the procedure continues automatically and a notification will appear in the wallet to allow the disclosure of the credentials. The second API is the POST /token. You can have a look to the swagger references here. The POST /token is required to exchange an authorization code (previously generated by the GET /auth). As a result it provides an access_token and an id_token, which contains the verifiable credentials revealed during authentication. These credentials contain information about the user.
Response Status Code	200 - OK

Table 8 – JWKS.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#FAAI-7
Endpoint Name	JWKS
Endpoint Description	JWKS endpoint containing the public keys used by OpenID connect relying party to verify any JWT issued by the authorization server.
Component	OIDC node
Endpoint URI	/oidc/jwks
HTTP Method	GET
Request (Query) Parameters	N/A
Request Body (example)	N/A
Request Headers	N/A
Response Body	{"keys":[{"e":"AQAB","n":"0Ycmwhk49Zqnh-nVRyTr-VICexhAklWihwu8YCvQWmASB_almhouFVzuJOtPN-3H2izmzge9zXl9jrDfijc2heOKb6VUogRVdI85FnrwNqWIrxmpzaweQkiCAjjlPmJ2-vIrNffB_cGtRgOauChonPY5cBAYfyzeR9b9eEx3r-R8p-ueu-TPaHH95gh7VIbDKUkGnFP0RFejjhP2vkfcQlhrCt1n8qWjiCrFLYRgeSI-iS2jomYxVMMwvupv1YEalkBeM2nmF0dtuV9ga7yfFZuqip5xC4Fg1oJENvJZ_XKkoQktxobvW4dMz-gbZvVeSQI0ZtjYMK2PqzzosM9BdQ","kty":"RSA","kid":"VpPJ0p_aILtDIhl3Jw7ZNqRJY_cALQAHe4rwbZJTWNE","use":"sig"},{"e":"AQAB","n":"moUHQzhodwqUt8g66nWjeHHeEo--OLg2o38JRVZqInUvQBd0YLroTE0cNcDgsvzGAOjobJ9phEtF_t9G5ppAnXRquLYltBv55ev6uTnGPVPUwb4WQPJhp12lKPhw1uTFlH6vNLK-D1UiaqSi0aE_IJLh1QgHFCp2MTyFgOmigPhfM3QGc_c8sSbDR

	SylY-SskMk9wU1vpMCjuD1gLcoxboljk2Dhe9uz2Mke-lhy0AFZdJ1K_xu8HkjLunjKMoz3Dtm1c917z3SyqA22l2Az1BeMyPXQaVSsTXCx48FHk8D9lCU1mT0K9ym5JQJxwGiTsapA6J5N4MkypWYEg7JECQ","kty":"RSA","kid":"aL062tC9RFzrqedyRxVR-G0TlZRpXPejpClNvLHvT4I","use":"sig"},{"crv":"P-256","x":"mf4VFYC5AxDVHCaGfwiCdLsbZlbOOdDu4bMLhL54Z9Y","y":"vW6edIo4q-ulhogfAgcCvp6Oa8Cp5_0HNxI2oaVLt6o","kty":"EC","kid":"XOAMa3HgSkc3MtREGUyCN2kjL19kbefx6xmqgnyQbws","use":"sig"},{"crv":"Ed25519","x":"yzGDb7GoVZ2aH8-owgGDlBbgmX8hHPMakSvo7fs9Se8","kty":"OKP","kid":"zR4lbLbFyFKIRMg4u-bwZ35Yr-hd-vXEKQ9BHjnPcos","use":"sig"}]}
Response Status Code	200 – OK

3.2 Assets Policy Manager (APM)

3.2.1 Description

The Assets Policy Manager (APM) plays a crucial role in ensuring the secure management of assets within FAME. This component serves two (2) key functionalities, both of which are vital for the smooth operation of FAME.

Firstly, the Assets Policy Manager enables the complete lifecycle management of policies associated with the federated assets discoverable through FAME, determining who is eligible to view and potentially acquire specific assets within FAME. Leveraging the Rule-Based Access Control (RBAC) model, the component considers various user and organizational attributes to combine them in Boolean expression formatted rules to make informed policy decisions. By fulfilling its role as a Policy Decision Point (PDP), the Assets Policy Manager ensures that other components of the project always display the appropriate assets to authenticated and authorized individuals or organizations. It acts as a central authority, facilitating the enforcement of access controls throughout the system.

Secondly, the component provides end-users with a comprehensive list of the assets that they have access to their contents. This encompasses assets uploaded by the end-users themselves or any other member of their organization (provenance) they are affiliated with, as well as assets acquired by them and have active contracts. This functionality offers the end-users a clear and consolidated view of their assets, enhancing their ability to manage and track their assets portfolio effectively. This functionality enhances transparency and accountability within the project, allowing users to have a clear view of the assets under their control.

Overall, the Assets Policy Manager plays a critical role within FAME, since it serves as the central authority for managing asset security policies and ensuring that the appropriate assets are displayed throughout the system. By acting as a PDP, it enables effective access control enforcement by other components. Additionally, the component provides the end-users with a comprehensive overview of their owned assets, enabling better asset management and control. With its pivotal functionalities, the Assets Policy Manager reinforces the project’s security posture, fosters transparency, and promotes efficient assets management practices.

3.2.2 Related Work

The APM has been developed in the context of the FAME project from the ground-up, capitalizing upon open-source technologies and tools, towards resolving a series of technical requirements as these have been documented in the FAME deliverable D2.1 (Requirements, Specifications and Co-Creation), and more specifically towards addressing:

- TR003: Be able to manage and enforce access and visibility restrictions on my assets, based on defined criteria (including e.g., organization type, user role, locality etc.) (also applicable also to TR110, TR207, TR607, TR715).
- TR902: Support access to the security policies of the underlying data marketplaces and data spaces.
- TR903: Support the consolidation of asset access policies at the level of the FAME federated asset space.
- TR904: Support the mapping of FAME policies to the lower-level policies of the underlying providers.
- TR005: be able to acquire and (locally) download discovered assets that are hosted in different marketplaces, data spaces etc. (also applicable also to TR120, TR602, TR610, TR710).
- TR005: Be able to acquire discovered assets.
- TR004: Be able to identify and discover assets that are hosted in different marketplaces.

3.2.3 Technical Specification

3.2.3.1 Component-level C4 Architecture

The Assets Policy Manager component consists of two (2) distinct modules: the Assets Policy Editor and the Assets Policy Engine. Each module plays a crucial role in enabling end users to define and enforce access policies for all the assets within FAME.

The first module, namely the Assets Policy Editor, empowers asset owners within FAME to define access policies that regulate who can view and potentially purchase their assets in the platform. This functionality is realized through three (3) levels of access: Confidential, Public, and Restricted. With Confidential access, only the owner of the asset can view it, ensuring utmost privacy and exclusivity. Public access allows all the authenticated users to view the asset, promoting open sharing and collaboration. Restricted access provides more granular control, allowing asset owners to define a set of and/or conditions based on user and organizational attributes (including but not limited to country, organization type, etc.), following the Rule-Based Access Control (RBAC) model. These conditions act as eligibility criteria for other users to meet for viewing the asset. The Assets Policy Editor incorporates a user-friendly UI to facilitate the intuitive definition of policies. Additionally, it provides a REST API that can be leveraged by the UI itself, as well as other components within the FAME platform, such as the Federated Data Assets Catalogue (FDAC) and Provenance & Tracing (P&T), to set policies during external asset indexing processes.

The second module, namely the Assets Policy Engine, acts as the Policy Decision Point (PDP) within the platform. It is responsible for answering two (2) fundamental questions related to asset's visibility and ownership for authenticated users. Firstly, it determines which assets a user can view within the platform, considering the defined access policies and user attributes. This ensures that users are presented with only the assets they are eligible to access. Secondly, the Assets Policy Engine provides information on the assets that the user has access to their contents. Content access includes assets that were uploaded by the user themselves or any other member of their organization (provenance) that the user is affiliated with, as well as assets that have been acquired with active contracts. To gather information about purchased assets, the Assets Policy Engine interacts with the Trading and Monetization module (T&M). Moreover, to extract the user and organization attributes that are

necessary to regulate asset's visibility, the Assets Policy Manager communicates with the Authentication & Authorization Infrastructure (AAI) to retrieve the required information before making the appropriate access decisions.

It is important to highlight that comprising the PDP of FAME, any component that needs to present users with assets information must first contact the Policy Engine to retrieve access eligibility details. Subsequently, these components will act as the Policy Enforcement Points (PEP) of the platform, either allowing or denying access based on the information received from the Policy Engine. This approach ensures consistent and appropriate enforcement of access policies throughout the platform, maintaining a secure and controlled environment for data asset management and utilization.

By encompassing both the Assets Policy Editor and the Assets Policy Engine, the Assets Policy Management ensures that assets' access within FAME is controlled, secure, and aligns with the defined policies. The collaboration between these components allows for user-friendly policy definition and efficient policy enforcement, guaranteeing that users are presented with only the relevant assets that they can access based on their attributes and ownership. The overall idea of the abovementioned approach is depicted in the 3rd level of the C4 diagram (i.e., component diagram) that is presented in the section before.

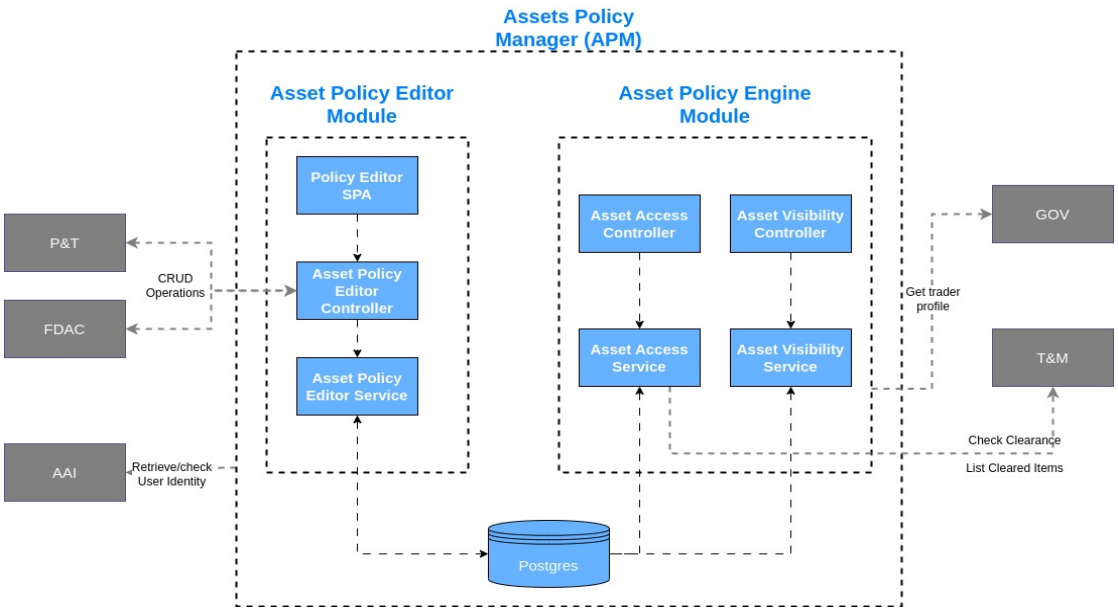

Figure 10 – 3rd level of the C4 diagram of the APM Component

3.2.3.2 Baseline Technologies and Tools

In terms of implementation, APM was built from the ground-up, capitalizing upon open-source technologies and tools. More specifically, the backend of the Assets Policy Editor and the Engine are based on Java 21 and the Spring Boot 3 framework is employed to facilitate efficient development and provide robust functionality. The data storage for the backend is handled by PostgreSQL (https://www.postgresql.org), a reliable and scalable open-source relational database management system. To enable policy definition and enforcement, both the Assets Policy Editor and the Assets Policy Engine utilize the EvalEx (https://github.com/ezylang/EvalEx) library. EvalEx is an expression evaluation library, allowing APM to efficiently define and evaluate complex rule-based expressions for the various permissions on the assets. Finally, regarding deployment, containerization solutions like Docker (https://www.docker.com) are employed to ensure consistent and reliable deployment across different environments.

3.2.4 Interfaces

To interact with the Asset Policy Manager, the **X-Identity** header should be provided for every request. This header will keep all the necessary information to identify the users and make decisions based on their profiles. To do so, the following JSON object should be provided as a base64 encoded string in the X-Identity header:

```
{
    "userId": "ABC",
    "organizationId": "ABC",
    "attributes": {
        "country": "Greece",
        "role": "Admin",
        "organizationName": "UBITECH",
        "organizationType": "SME",
        ǂ ł
        ǂ
    }
}
```

The structure of the JSON contains the following info:

- userId: The unique identifier of the user
- organizationId: The unique identifier of the organization that the user belongs to
- attributes: A nested object containing additional attributes related to the user and/or the organization they belong to. This is useful to make decisions regarding what assets the users can view in the marketplace (by also combining info from the T&M module). This object can contain any other attributes that we would like to include, or even fewer than the ones stated above. However, the attributes mentioned above are a good starting point.

Table 9 – APM addPolicy Technical Interface.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#APM1
Endpoint Name	addPolicy
Endpoint Description	This endpoint is responsible for exposing the necessary functionality to create a new policy for an asset
Component	APM
Endpoint URI	/api/v1/asset-policy-editor
HTTP Method	POST
Request (Query) Parameters	N/A

Request Body (example)	{
	"assetType": "DATASET",
	"assetId": "f6f94402-fad0-4fed-8df5-60dd46831170",
	"accessType": "RESTRICTED",
	"rule": "(country == \"Greece\")"
	}

Request Headers	X-Identity: a custom header to accept the user profile in a Base64 encoded format.
Response Body	{ "id": 1 "assetType": "DATASET", "assetId": "f6f94402-fad0-4fed-8df5-60dd46831170", "accessType": "RESTRICTED",

	"rule": "(country == \"Greece\")" }
Response Status Code	201 - Created

Table 10 – APM getPolicy Technical Interface.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#APM2
Endpoint Name	getPolicy
Endpoint Description	This endpoint is responsible for fetching an existing policy
Component	APM
Endpoint URI	/api/v1/asset-policy-editor
HTTP Method	GET
Request (Query) Parameters	assetId: The ID of the asset to retrieve the policy for
Request Body (example)	N/A
Request Headers	X-Identity: a custom header to accept the user profile in a Base64 encoded format.

Response Body	{
	"id": 1
	"assetType": "DATASET",
	"assetId": "f6f94402-fad0-4fed-8df5-60dd46831170",
	"accessType": "RESTRICTED",
	"rule": "(country == \"Greece\")"
	}

Response Status Code	200 - OK

Table 11 – APM updatePolicy Technical Interface.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#APM3
Endpoint Name	updatePolicy
Endpoint Description	This endpoint is responsible for updating an existing policy of a specific asset
Component	APM
Endpoint URI	/api/v1/asset-policy-editor
HTTP Method	PUT
Request (Query) Parameters	N/A

Request Body (example)	{
	"assetType": "FILE",
	"assetId": "f6f94402-fad0-4fed-8df5-60dd46831170",
	"accessType": "RESTRICTED",
	"rule": "(country == \"Greece\" && organizationType == \"SME\")"
	}

Request Headers	X-Identity: a custom header to accept the user profile in a Base64 encoded format.
Response Body	N/A
Response Status Code	204 – No Content

Table 12 – APM deletePolicy Technical Interface.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#APM4
Endpoint Name	deletePolicy
Endpoint Description	This endpoint is responsible for deleting a policy for an existing asset
Component	APM
Endpoint URI	/api/v1/asset-policy-editor
HTTP Method	DELETE
Request (Query) Parameters	assetId: The ID of the asset to delete the policy for
Request Body (example)	N/A
Request Headers	X-Identity: a custom header to accept the user profile in a Base64 encoded format.
Response Body	N/A
Response Status Code	204 – No Content

Table 13 – APM contentAccessCheckOne Technical Interface.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#APM5
Endpoint Name	contentAccessCheckOne
Endpoint Description	This endpoint is responsible for checking if the underlying user has content access to a specific asset
Component	APM
Endpoint URI	/api/v1/asset-access/check-one
HTTP Method	GET
Request (Query) Parameters	assetId: The ID of the asset to check for content access
Request Body (example)	N/A
Request Headers	X-Identity: a custom header to accept the user profile in a Base64 encoded format.

Response Body	{
	"hasAccess": true,
	"assetAccessType": "OWN"
	}
	
	{
	"hasAccess": true,
	"assetAccessType": "BOUGHT"
	}
	
	{
	"hasAccess": false
	}

Response Status Code	200 - OK

Table 14 – APM contentAccessCheckMany Technical Interface.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#APM6
Endpoint Name	contentAccessCheckMany
Endpoint Description	This endpoint is responsible for checking if the underlying user has content access to a list of assets.
Component	APM
Endpoint URI	/api/v1/asset-policy-editor/check-many
HTTP Method	GET
Request (Query) Parameters	assetId: The ID of the asset to delete the policy for
Request Body (example)	A list of asset IDs ["db12fdba-cb79-4867-8642-896bf297374d", "a09643c2-8019-4b68-9d19-85bc3c3c7807", "2790dbeb-90e9-413c-aeda-605c44320f09"]
Request Headers	X-Identity: a custom header to accept the user profile in a Base64 encoded format.

Response Body	[
	{
	"hasAccess": true,
	"assetAccessType": " OWN "
	},
	{
	"hasAccess": true,
	"assetAccessType": "OWN"
	},
	{
	"hasAccess": false
	}
	]

Response Status Code	200 – OK

Table 15 – APM contentAccessCheckAll Technical Interface.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#APM7
Endpoint Name	contentAccessCheckAll
Endpoint Description	This endpoint is responsible for retrieving a list of all the assets that the underlying user has access to their contents.
Component	APM
Endpoint URI	/api/v1/asset-access/check-all
HTTP Method	GET
Request (Query) Parameters	assetType (OPTIONAL): Limit results based on a specific asset type.
Request Body (example)	N/A

Request Headers	X-Identity: a custom header to accept the user profile in a Base64 encoded format.

Response Body	{
	"own": [
	db12fdba-cb79-4867-8642-896bf297374d,
	a09643c2-8019-4b68-9d19-85bc3c3c7807
	],
	"bought ": [
	2790dbeb-90e9-413c-aeda-605c44320f09
	]
	}

Response Status Code	200 – OK

Table 16 – APM marketplaceVisibilityCheckOne Technical Interface.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#APM8
Endpoint Name	marketplaceVisibilityCheckOne
Endpoint Description	This endpoint is responsible for checking if the underlying user can view a specific asset on the Marketplace
Component	APM
Endpoint URI	/api/v1/asset-visibility/check-one
HTTP Method	GET
Request (Query) Parameters	assetId: The ID of the asset to check for marketplace visibility
Request Body (example)	N/A
Request Headers	X-Identity: a custom header to accept the user profile in a Base64 encoded format.

Response Body	{
	"hasVisibility": false
	}

Response Status Code	200 - OK

Table 17 – APM marketplaceVisibilityCheckMany Technical Interface.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#APM9
Endpoint Name	marketplaceVisibilityCheckMany
Endpoint Description	This endpoint is responsible for checking if the underlying user can view a list of assets on the marketplace
Component	APM
Endpoint URI	/api/v1/asset-visibility/check-many
HTTP Method	GET
Request (Query) Parameters	N/A
Request Body (example)	["db12fdba-cb79-4867-8642-896bf297374d", "a09643c2-8019-4b68-9d19-85bc3c3c7807"]

Request Headers	X-Identity: a custom header to accept the user profile in a Base64 encoded format.

Response Body	[
	{
	"hasVisibility": false
	},
	{
	"hasVisbility": true
	}
	]

Response Status Code	200 - OK

Table 18 – APM marketplaceVisibilityCheckAll Technical Interface.

FAME TECHNICAL INTERFACE	
Technical Interface ID	#APM10
Endpoint Name	marketplaceVisibilityCheckAll
Endpoint Description	This endpoint is responsible for retrieving a list of all the asset IDs that the underlying user can view on the marketplace
Component	APM
Endpoint URI	/api/v1/asset-visibility/check-all
HTTP Method	GET
Request (Query) Parameters	assetType (OPTIONAL): Limit results based on a specific asset type.
Request Body (example)	N/A
Request Headers	X-Identity: a custom header to accept the user profile in a Base64 encoded format.
Response Body	["db12fdba-cb79-4867-8642-896bf297374d", "a09643c2-8019-4b68-9d19-85bc3c3c7807"]
Response Status Code	200 - OK

The above endpoints are also included in detail on the OpenAPI specification format which is available on the projects Gitlab at the following URL:

https://gitlab.infinitech - h2020.eu/fame/framework/apm/ - /blob/master/OpenAPI.yaml

4 Components Demonstration

4.1 Federated Authentication and Authorization Infrastructure (FAAID)

The verifiable credential (VC) service manages secured credentials, allowing through APIs to issue, verify, revoke verifiable credentials in the W3C standard. Since authentication and authorization takes place through the selective detection of user attributes (i.e., W3C verifiable credentials https://www.w3.org/TR/vc-data-model/#what-is-a-verifiable-credential), a service is therefore necessary that allows the issuance, revocation and verification.

This service acts as an "issuer" role in the self-sovereign identity model. Read more at W3C online Recommendation here: https://www.w3.org/TR/vc-data-model/#ecosystem-overview

Unlike traditional OIDC (OpenID Connect) providers that store user data in a centralized database, FAME's user management adheres to the self-sovereign identity model. In this approach, users retain exclusive control of their information within wallet applications. Authentication and authorization processes are conducted through the selective disclosure of specific user attributes.

4.1.1 Prerequisites and Installation Environment

The Verifiable credentials service is available in the form of a Docker image, can be deployed in any environment (local, cloud) and in any operating systems (Windows, Mac OS, Linux).

The following programs and libraries are required to be able to perform all subsequent instructions.

- WSL
- Git
- Node.js
- Docker
- FAME wallet desktop

To check that everything is fine, you should see the following commands working:

```
$ npm -v
6.14.8
$ node -v
v14.15.1
$ ts -node -v
v10.4.0
$ nvm -- version
0.33.2
```

WARNING: In case some packages are missing, you can install them using the:

```
npm i <PACKAGE_NAME>Command
```

4.1.2 Installation Guide

Find below a step-by-step document detailing how the current component can be locally installed by any user.

4.1.2.1 Initial environment configuration

Create a .env.vc and a .env file and configure the service using the environmental variables defined in the template.env file. WARNING: Paths should be relative to the app directory

You can run the project in development in two ways: directly through Node.js or using Docker. Both modes are described below.

4.1.2.2 Initial database configuration for OIDC service

A just-working version using a mongodb can be used. Before starting the mongo for the first time, copy db/scripts/mongo-init.template.js to db/scripts/mongo-init.js and update to your needs. The contents of that file are the initial state of the DB. By default, it comes with to RP clients, one for WebApps and another for Native Apps or SPAs; as well as an administration account that can be used to create more clients through the API.

4.1.2.3 Local development using Node.js

To run the service locally using Node.js it is necessary to download it before. After that you can install the dependencies and start the service in the following way:

```
$ cd verifiable - credentials/app or cd node - oidc - provider/app
$ npm i
$ npm start
```

You have also to update the configuration file app/src/config.ts before running the service. Specifically, it is necessary to fill the default environment variables, in the same way they are filled in the env file.

4.1.2.4 Local development using docker

To start-ups the project in development way using docker you must run the following command in the project root. The first time it will take a while (be patience), since it has to build images and download all the npm dependencies.

```
./docker - dev- start
```

The OAS documentation can be accessed on http://localhost:4200/release2/vc/api-spec/ui.

You can stop the container at any time with Ctrl-C.

If you want to delete and prune all the created images, containers, networks, volumes, just run.

```
./docker - dev- prune
```

Since the app directory is shared with the docker container with mapped user permissions, you can just edit any files in the app directory locally. The container will be running ts-node and nodemon to directly execute the source code and refresh the server if any file has changed. You can also attach any debugger in your local machine to the container, which will be listening at default port 9229.

4.1.2.5 Development scripts in the docker container

Since npm and node are likely to be needed, if your OS allows you to execute shell scripts inside the docker container, you can just also use the npm and node scripts provided for convenience.

```
$ ./npm - v
6.14.8
$ ./node - v
v14.15.1
```

4.1.2.6 Building the production image

You can build the production docker image using the helpers provided in this repository:

Build the image to work locally!

```
./docker - prod - build
```

Build and push the image into the GitLab registry.

```
./docker - prod - push
```

The script docker-prod manages the deployment. You can use it to extract the files required in the production environment. To do so execute the following commands:

```
# Prepare a folder to work
mkdir oidc - provider
# Place the 'docker - prod' file inside oidc - provider
cd oidc - provider
# Pull the docker image and prepare the volumes
./docker - prod - p init - volumes $(id - u):$(id - g)
# Create a docker - compose.yaml and .env using the template.
# Remember to configure SERVER_PUBLIC_URI, OIDC_PROVIDER_DB_PASSWORD and
MONGO_INITDB_ROOT_PASSWORD in the .env file
./docker - prod - o docker - compose.yaml template docker - compose
./docker - prod - o .env template env
# Start docker services
docker - compose up - d
# We have an issue on the first start. If you get an error just restart docker services
docker - compose down
docker - compose up - d
```

4.1.2.7 Production deployment

To deploy this service in a server it is just necessary to copy the docker-compose.yaml and vc.env files in a server directory and run the following command:

```
$ docker - compose up
```

The docker-compose will pull the production image directly from the FAME Gitlab container registry. You need to have a valid Gitlab access to pull it.

4.1.3 User Guide

This API is deployed and accessible from this URL.

https://identity.fame - horizon.eu/release2/vc/api - spec/ui/ .

The following module defines how to use this service.

4.1.3.1 Issue a Credential

This APIs produces an HTML page that must be displayed on a browser. It is therefore necessary to redirect the user at the URL of this API. The generated HTML page contains a script that communicates with the fame wallet, which must be running on *localhost:8000*. Following the rendering of this page, a notification will appear on the wallet asking you to select the identity (DID) on which to save the verifiable credential. After selecting it, it automatically generates the credential and sends it back to the FAME wallet. The wallet will present a new notification asking if you want to accept the credential. Upon acceptance, the credential will be saved and viewable in the wallet.

Endpoint:

```
GET/ issue /{credential}/ callbackUrl /{callbackUrl}
```

- The {credential} parameter is a JSON that represent the verifiable credential payload encoded as a URL. An example of a credential's payload can be:

```
{
"consumer": "true"
}
```

- You can encode this payload as a url in this way `encodeURIComponent(JSON.stringify({ consumer: true }))`, obtaining the string `%7B%22consumer%22%3Atrue%7D` that is has to be passed as parameter.
- The {callbackUrl} parameter is a URL encoded as a URL. E.g., `https://i3m - marketplace.eu` will become `%22https%3A%2F%2Fi3m-marketplace.eu%22`.

4.1.3.2 Revoke a credential

This API takes a verifiable credential as input in the body, calculates the hash and registers it in the credential registry, on the FAME Besu blockchain.

Endpoint:

```
POST/ revoke
```

With the following body

```
{
   "credentialJwt": "string"
}
```

Swagger references here:

https://identity.fame - horizon.eu/release2/vc/api - spec/ui/#/Credential/post_credential_revoke

4.1.3.3 Verify a credential

This API checks whether a credential in JWT format is registered in the revocation registry. The JWT of the verifiable credential and optionally the DID of the revoker you want to verify must be passed in the body. If the credential Issuer, i.e., the revoker, it is not specified, it will be checked whether the issuer of the verifiable credential has revoked the credential or not. This way you can check if the credential has been revoked from a specific DID or not.

Endpoint:

```
POST/ verify
```

With the following body

```
{
   "credentialJwt":        "string"
   "credentialIssuer": "string"
}
```

Swagger references here:

```
https://identity.fame        - horizon.eu/release2/vc/api        - spec/ui/#/Credential/post_credential_verify
```

4.2 Federated Assets Policy Manager (FAPM)

4.2.1 Prerequisites and Installation Environment

The Assets Policy Manager is available in the form of a Docker image. APM can be deployed in any environment (local, cloud) and in any operating system (Windows, Mac OS, Linux), as long as Docker and docker-compose plugin are installed. The Docker image is available on project's Harbor, which is a container registry, at the following URL:

```
https://harbor.infinitech        - h2020.eu/harbor/projects/40/repositories/framework%2Fapm
```

4.2.2 Installation Guide

To deploy APM in any environment, you can utilize the provided docker-compose.yaml file which is available on the projects Gitlab at the following URL:

```
https://gitlab.infinitech        - h2020.eu/fame/framework/apm
```

As long as this file is available and Docker alongside with docker-compose are installed, you can run the **docker-compose up –d** command to start up the APM. The specific installation assumes that ports 5432 and 8080 are not being already used by another process. If any of those are already occupied, one should first modify the exposed ports on the compose file. Furthermore, the default image of APM that is used is a GraalVM Native compiled one. Native images provide significant performance boost and resource utilization compared to typical images. However, some hardware architectures do not support native images and it might be the case that during the component start-up, the process will fail. If such a problem is encountered, modify the image name in the compose file and switch from the Native image to the plain one, since both image versions are available.

4.2.3 User Guide

A detailed description of the use of the APM component is available in the form of a reference video, which is currently hosted in the project repository and is available to the project partners. Once the FAME Learning Hub matures, the reference video will also be made available through the FAME Learning Hub.

5 Conclusions

FAME aims to establish a network of Data Marketplaces, facilitating efficient data exchange between Data Consumers and Providers while fostering the emergence of new data services. The primary objectives include ensuring a requisite level of trust and upholding privacy principles. To achieve this, all clients within the network must authenticate FAME users, who, in turn, must be uniquely identified across the entire network.

FAME is committed to incorporating trust and privacy "by design" through features such as data minimization, explicit user consent, limitations on purposes, data accuracy, integrity, and confidentiality. The inadequacies of current centralized and federated models in addressing integration, trust, and privacy challenges necessitate an alternative approach.

Consequently, FAME has chosen to adopt a self-sovereign identity management model, leveraging emerging digital identity solutions that utilize distributed ledger and cryptography technologies. This model implements a user-centric authentication approach while adhering to privacy principles "by design". Utilizing Distributed Identifiers (DID) and Verifiable Credentials (VC), FAME users can create and manage their identities, exert control over their personal data and credentials, and selectively share them with explicit consent, limiting exposure to only necessary information. In this framework, FAME users are uniquely identified by DIDs, and the VCs shared by users can be readily verified by third-party applications without involving the credential issuer, thereby circumventing integration challenges among FAME stakeholders.

Within the innovative framework of FAME, the Assets Policy Manager (APM) emerges as an essential component, seamlessly integrating with the network's core objectives of establishing a secure and privacy-centric network of Data Marketplaces. The APM's functionality is twofold and pivotal in enhancing FAME's data exchange ecosystem. Firstly, it administers the complete lifecycle management of asset-related policies, ensuring that only authenticated and authorized users, as defined by FAME's self-sovereign identity management model, can access specific assets. Secondly, the APM provides a comprehensive and transparent view of the assets to end-users, thereby reinforcing FAME's commitment to user-centric data management and privacy. In addition, in the context of safeguarding security and privacy of the assets handled by FAME, it incorporates an Assets Policy Manager (APM), that plays a crucial role in ensuring the secure management of assets within it. The Assets Policy Manager leverages the Rule-Based Access Control (RBAC) model and enables the complete lifecycle management of policies associated with the federated assets discoverable through FAME, determining who is eligible to view and potentially acquire specific assets within FAME. The APM acts as a Policy Decision Point (PDP), facilitating the enforcement of access controls throughout the system. In addition, the APM provides end-users with a comprehensive list of the assets that they have access to their contents, offering the end-users a clear and consolidated view of their assets, enhancing their ability to manage and track their assets portfolio effectively. The Assets Policy Manager is available in the form of a Docker image and can be deployed in any environment (local, cloud) and in any operating system.

Overall, the inclusion of the Assets Policy Manager in FAME's architecture exemplifies the synergy between secure asset management and the network's overarching goals of fostering efficient data exchange, ensuring trust, and upholding stringent privacy standards.

Table 19 – D3.1 Related KPIs.

Obj. ID	KPI ID	Measured KPI	Expected Value [*M9*]	Achieved Value [*M9*]	Expected Value [*M18*]	Achieved Value [*M18*]
O2	**2.1**	Data Marketplaces/Spaces implementing the Data Provider Interface	0	0	3	N/A
O2	**2.2**	Security and Data Protection Policies to be abstracted	6	11	13	N/A
O2	**2.3**	Increased Prevention of Policy Violations on Data Assets from other marketplaces	13%	100%	25%	N/A

6 References

1. W3C Credentials Community Group. (n.d.), "Decentralized Identifiers (DIDs) - W3C Working Draft.," [Online]. Available: https://w3c-ccg.github.io/did-spec.
2. World Wide Web Consortium, “Verifiable Claims Working Group”, 2017. [Online]. Available: https://www.w3.org/2017/vc.
3. A. Mavrogiorgou, et al., "FAME: Federated Decentralized Trusted Data Marketplace for Embedded Finance", [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10215814.
4. FAME, "D2.1 - Requirements Analysis, Specifications and Co-Creation I," 2023.
5. World Wide Web Consortium (n.d), "Verifiable Credentials Data Model," [Online]. Available: https://www.w3.org/TR/vc-data-model-2.0/.
6. OpenID Foundation (n.d.), "OpenID for Verifiable Credentials," [Online]. Available: https://openid.net/sg/openid4vc/.
7. OpenID Foundation. (n.d.), [Online]. Available: https://openid.net/.
8. T. Lodderstedt, T. Yasuda and T. Looker, "OpenID for Verifiable Credential Issuance," 26 November 2023. [Online]. Available: https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html.
9. i3-Market , [Online]. Available: https://www.i3-market.eu/i3-market-architecture/.
10. Veramo, “Veramo, a decentralized identity and credential management library for JavaScript/TypeScript”, [Online]. Available: https://veramo.io/.
11. Uport, “Uport library”, [Online]. Available: https://developer.uport.me/ethr-did/docs/index.
12. F. Hauck, *OpenID for Verifiable Credentials*, 2023.
13. N. Sakimura, J. Bradley and N. Agarwal, “Proof Key for Code Exchange by OAuth Public Clients”, September 2015. [Online]. Available: https://datatracker.ietf.org/doc/rfc7636/.