Federated decentralized trusted dAta Marketplace for Embedded finance

D4.3 - Operational Models, Business Models and Governance I

Title	D4.3 - Operational Models, Business Models and Governance I
Revision Number	1.0
Task reference	T4.5
Lead Beneficiary	JRC
Responsible	Konstantina Tripodi
Partners	ENG, EUBA, JOT, TRB, UIA
Deliverable Type	DEM
Dissemination Level	PU
Due Date	2024-04-30 [Month 16]
Delivered Date	2024-05-31
Internal Reviewers	UBI GFT
Quality Assurance	UPRC
Acceptance	Coordinator Accepted
Project Title	FAME - Federated decentralized trusted dAta Marketplace for Embedded finance
Grant Agreement No.	101092639
EC Project Officer	Stefano Bertolo
Programme	HORIZON-CL4-2022-DATA-01-04

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

Revision History

Version	Date	Partners	Description
0.1	2024-02-01	JRC	Table of Contents
0.2	2024-03-02	JRC, ENG, EUBA, JOT, TRB, UIA	Integrated version with contributions
0.3	2024-04-02	JRC, ENG, EUBA, JOT, TRB, UIA	Updated version with contributions from all
0.5	2024-04-19	JRC, ENG, EUBA, JOT, TRB, UIA	Updated version with new contributions
0.6	2024-04-28	JRC, ENG, EUBA, JOT, TRB, UIA	Updated and upgraded version
0.7	2024-05-20	JRC, ENG, EUBA, JOT, TRB, UIA	Version for peer review
0.8	2024-05-23	JRC, ENG, EUBA, JOT, TRB, UIA	Update after peer review
0.9	2024-05-30	JRC, ENG, EUBA, JOT, TRB, UIA	Version for QA
1.0	2024-05-31	JRC	Version for submission

Definitions

Acronyms	Definition
AAI	authentication authorization infrastructure
ACM	Association for Computing Machinery
API	Application Programming Interface
AWS	Amazon Web Services
CAN	Campus Area Network|Controller Area Network
ERC	Ethereum Request for Comments
EU	European Union
FAME	Federated decentralized trusted dAta Marketplace for Embedded finance
FDAC	Federated Data Assets Catalogue
GDPR	General Data Protection Regulation
GOV	Operational Governance
HTTP	HyperText Transfer Protocol
ID	Identity
IEEE	Institute (of) Electrical (and) Electronic Engineers
IP	Internet Protocol
ISO	International Organization for Standardization
IT	Information Technology
JSON	JavaScript Object Notation
MVP	Minimum Viable Product|Platform
OS	Operating System
PDF	Portable Document Format (for Adobe Acrobat Reader)
RA	Reference Architecture
REST	Representational State Transfer
RPC	Remote Procedure Call
SA	Supervisory Authority
SDK	Software Development Kit
SLA	Service Level Agreement
SME	Small and Medium-Sized Enterprises
SQL	Structured Query Language
SSI	Server Side Includes
TID	Trade Account Identifier
TXT	Text|TeXT file
UID	User Identifier
USD	United States Dollar

Executive Summary

This deliverable D4.3 “Operational Models, Business Models and Governance” is the first tangible results of the effort of task T4.5 “Business and Operational Models for the FAME Marketplace”. It outlines the business models and minimum viable product configuration for the Operational Governance (GOV) module of the FAME Marketplace. The GOV module aims to provide essential services for user registration and support various business models like pay-as-you-go and data-as-a-service, as well as trading mechanisms.

The objective of this deliverable is to identify and analyze the optimal mix of business models for monetizing the FAME Marketplace and describe the governance models for its operations. Unlike previous configurations, however, SLA management now falls under Task T4.1.

In our exploration, we evaluated 18 business models using a Business Models Selection Matrix and refined the selection through pilot validation, as selecting the right business models is crucial for ensuring the financial viability of the marketplace and aligning with pilot needs. The final selection includes subscription-based models, pay-as-you-go (PAYG), pay-as-you-use (PAYU), Data-as-a-Service (DaaS), and freemium models. These models are designed to maximize marketplace value and pave the way for commercialization.

The GOV module acts as the gateway for all essential management and governance functions of the FAME federation. It comprises four components:

- Federation Management: for administering federation members, including onboarding, offboarding, records management, and membership checks.
- User Management: for managing the platform’s user pool.
- User Support (newly added): for managing ticketing systems and user request queues.
- Trading Support: for managing trading account permissions, linking trading parties, offering insights and historical data, and managing token exchange services.

These components form the minimum viable product (MVP) of the GOV module, enabling the FAME Platform to onboard new members, authenticate users, grant administrative privileges, authorize blockchain accounts for trading, and process user requests asynchronously.

In the next project phase, we will focus on refining and implementing the selected business models and governance structures. Key activities include incorporating user interaction metrics to enhance the pricing advisory mechanism, expanding GOV module functionalities, and continuous pilot validation to adapt the business models and operational framework based on feedback. This iterative process ensures that the FAME Marketplace remains adaptive and responsive to user needs and market dynamics.

Table of Contents

List of Figures

List of Tables

1 Introduction

This deliverable will present the business models of the FAME marketplace and its minimum viable product configuration for the Operational Governance (GOV) platform module. The goal of the Operational Governance component is to provide the necessary services for users' registration, and the support of business models such as pay-as-you-go and data-as-a-service, in subscription management, as well as trading mechanisms. Differently from previous information, SLA management has been moved under the scope of task T4.1.

The Business Models selection is key to ensure financial viability of the FAME marketplace as well as the maximization of alignment among Pilots. Therefore, the document will depict the FAME's methodology to identify such models, suggesting a hybrid approach to satisfy all the needs while providing the foundations for effective monetization schemes.

The GOV – which includes both the operational models and governance frameworks – is the blueprint that defines how the marketplace functions on a day-to-day basis. It encompasses processes, workflows, and technology infrastructure. Our operational model for FAME Marketplace focuses on efficiency, scalability, and user experience. Ethical and responsible governance is at the core of FAME Marketplace, establishing guidelines for a transparent and accountable governance structure ensures that all participants, including buyers, sellers, and our internal teams, adhere to ethical standards.

1.1 Objective of the Deliverable

This deliverable aims at analyzing and identifying the optimal mix of business models to monetize the FAME marketplace successfully, as well as to describe the governance models for the operations of the marketplace including the four main components: (i) administration of federation members, (ii) user management, (iii) user support, (iv) trading transactions management. The proposed deliverable is the first one under the scope of Specification and implementation of T4.5 – "Business and Operational Models for the FAME Marketplace" that aims to strategically exploit the federated marketplace model, capitalizing on its unique structure and collaborative ecosystem. Our approach encompasses fostering collaborative partnerships with synergistic businesses within the marketplace, leveraging data insights for informed decision-making and potential monetization, and ensuring a seamless, unified user experience. This deliverable positions the business for sustained success within the dynamic landscape of the federated marketplace.

1.2 Insights from other Tasks and Deliverables

T4.5 works closely with the other tasks in WP4, namely: T4.1 "Decentralized Data Provenance and Traceability", for initial inputs, T4.2 "Accounting, Trading, Pricing and Monetization Schemes", for trading and monetization mechanisms, T4.3 "Smart Contracts for Programmable Trading and Monetization", for smart contracts and trading, and T4.4 "Semantic Search for Trading and Valuation of Data Assets", for semantic search over the federated catalogue. Other insights are from T3.1 "Federated AAI Infrastructure", and T3.2 "Unified Security Policy Management", for authentication and authorization components. Lastly, this deliverable will serve as input for T7.3 "Exploitation and Sustainability Planning", to leverage the business models of the FAME Marketplace.

1.3 Structure

The document contains six (6) chapters:

1. *Introduction*: Provides an initial introduction, overview of the objectives, insights from related tasks, and the structure of the deliverable.

2. *Data Marketplace Business Models*: Discusses the various business models considered, their evaluation, and the selection process for the most suitable models.
3. *Module Overview*: Describes the components and architecture of the FAME Marketplace, detailing how different modules integrate and operate.
4. *Components Specification*: Offers technical documentation for the APIs and integration points, detailing the functionalities and technical requirements of each component.
5. *Components Demonstration*: Showcases the implementation and deployment of the MVP (Minimum Viable Product) demonstrator, including custom deployment guides for server-based and blockchain-based software.
6. *Conclusions*: Summarizes the achievements of the deliverable, the operational and business models established, and outlines future work and enhancements.

2 Data Marketplace Business Models

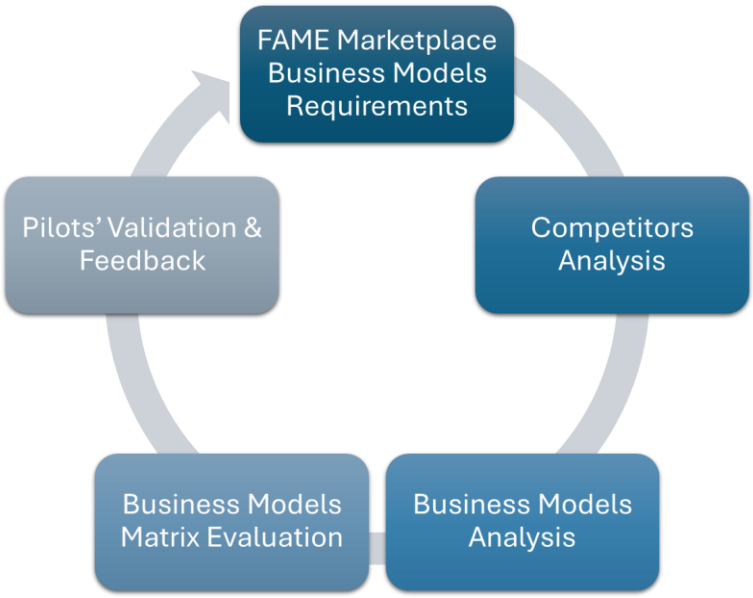

Figure 1 - Business Models Framework

FAME will explore various business models to ensure the maximization of value created by the marketplace, leveraging on early adopters as well as monetization schemes. Various models will be analyzed through a framework to develop the best fitting combination, tailored to the project's objectives. The marketplace will allow stakeholders to integrate configurable, transparent and decentralized monetization and market dynamics schemes, enabled by the selected business models to maximize the value added to both data sellers and buyers, enhanced by the various trading and pricing schemes.

The following chapters will compare FAME with competitors' marketplaces, highlighting the unique value proposition and deepening on the trading and pricing schemes. Consequently, an exhaustive list of business models will be presented, analyzed using the Business Models Matrix, and finalized to select the best 3 models. Finally, the pilots' validation questionnaire will be shown, detailing the feedback provided to further enhance the business models combination for the FAME marketplace.

2.1 Competitor Analysis

The major work by [1] provides a survey of data marketplaces and their business models. The work by [2] provides analysis of the prices of different categories of data. [3] also presents a survey of pricing or data marketplaces. These three works present invaluable and comprehensive study of the state-of-the-art state of data marketplaces. Their results can be utilized while performing competitor analysis. Table 3 lists a number of characteristics of competitor marketplaces against FAME.

A sell-side market facilitates selling of data from multiple data providers to data shoppers. A buy-side market makes it possible for individuals and organizations to capitalize on their data by enabling data dealers to acquire data from them. A two-sided market combines both sell-side and buy-side mechanisms. FAME can be categorized as two-sided market where data assets are listed by data providers (sellers) in a catalog and individuals and organizations (which can themselves be sellers) can purchase data via blockchain based payment transactions.

A datamarket can provide (i) open data repositories (ii) generalized purpose (iii) specialized (niche) data. Since FAME is a federated marketplace, it can provide all these three categories of data. Open

data repositories can help attract customers to the data providers. Different niche data providers can be present under the same FAME market umbrella.

Most of the marketplaces are traditionally centralized. Whereas centralized markets can offer advantages like higher levels of privacy, efficiency and more responsive user interfaces, they are less flexible in terms of transactional autonomy. In financial contexts, transactional autonomy refers to the ability of individuals or organizations to carry out financial transactions, such as buying or selling assets, without requiring explicit approval or involvement from a central authority. Amazon Marketplace, for example, handles billing, collections, and disbursement of payments and hence we can say the data provider does not have transactional autonomy; Amazon Marketplace carries out transactions and also charges a commission called referral fee. Since FAME is blockchain based, data providers have transactional autonomy: they can carry out payment transactions using tokenized digital currency directly with their customers. What is more, they can also employ dynamic pricing mechanisms. FAME provides smart contract-based governance bourse operations that support ERC20, ERC721 and ER1155 token standards. Hence, support for additional currency tokens as well as utility tokens issued by data providers exists and can readily be deployed on the FAME blockchain with negligible costs. Note that transaction transparency is also higher on blockchains. Whereas, this may be disadvantageous in terms of privacy, it can be advantageous from the perspective that it can contribute to market efficiencies (the degree to which prices in financial markets reflect all available information) and achievement of fair pricing.

Companies like Amazon and Snowflake also offer compute and storage services in addition to data marketplace. As a result, a customer can combine these services, i.e. buy data and use it in a computation server, all in the same infrastructure. The ability to couple data, computation and storage services offers big advantages, especially if the data purchased is big data. Data can be processed in the same place where it is stored and sold, hence, providing high efficiency and performance. FAME does not offer storage and compute services.

Note that FAME market that runs on the blockchain can act as an embedded finance solution, since it provides a bridge between digital tokenized currencies and the traditional fiat money gateways. Via Metamask wallet, it enables users to make payments for data assets.

Table 1 - Comparison of FAME with other competitor marketplaces

	Amazon AWS Data Exchange	*Snowflake Data Marketplace*	*Databricks Marketplace*	*Datum*	*AggData*	*Data and Sons*	*FAME*
Sell side market	Yes	Yes	Yes		Yes	Yes	
Buy side market						Yes	
Two-sided market						Yes	Yes
Open data repositories	Yes					Yes	Yes
General purpose data	Yes	Yes	Yes			Yes	Yes
Specialized (Niche) Data	Yes	Yes	Yes		Yes (Locational)	Yes	Yes
Centralized	Yes	Yes			Yes	Yes	
Blockchain based decentralization				Yes			Yes
Transactional Autonomy							Yes
Tokenized digital currency payments							Yes
Utility Token payments				Yes			Yes

Credits or Units		Credits	DBU				
Dynamic pricing tools							Yes
Computation Service	Yes	Yes	Yes	Yes			
Storage Service	Yes	Yes	Yes	Yes			
Privacy	Higher	High	High	High	High	High	Lower
Transaction Transparency	Lower	Lower				Lower	Higher
Embedded Finance							Yes

In addition to these marketplaces, there are a range of platforms, tools, and protocols that engage in decentralized data management, trading, storage, and related services. The focus of the following paragraphs is to explain their core functionalities, value propositions, and their role in the ecosystem of decentralized technologies.

1. Ocean Protocol: Ocean Protocol offers a decentralized framework for data sharing and monetization. It leverages blockchain technology to allow individuals and organizations to share and monetize data while ensuring privacy and control.

- Value Proposition: The protocol supports data wallets, marketplaces, and compute-to-data functionalities, enabling data providers to maintain privacy and control.

2. Streamr: Streamr provides a real-time data trading ecosystem where users can monetize and share data streams in a decentralized manner.

- Value Proposition: The platform integrates a decentralized pub/sub network that facilitates the seamless exchange of information in real-time.

3. Numerai: Numerai operates as a decentralized hedge fund, utilizing a global network of data scientists who contribute predictive financial models.

- Value Proposition: It uniquely combines machine learning and blockchain to crowdsource predictive models, rewarding contributors with its native cryptocurrency.

4. Fetch.ai: Fetch.ai is developing a decentralized machine learning platform focused on creating autonomous economic agents that can perform proactive tasks.

- Value Proposition: It emphasizes autonomous agent systems that operate independently, making decisions and transacting without human intervention.

5. Enigma: Focused on privacy-enhancing technologies, Enigma provides solutions for secure and private data sharing and computation.

- Value Proposition: Enigma's main feature is its ability to enable private computations on data while it remains encrypted, fostering trustless sharing of sensitive data.

6. NuCypher: NuCypher presents cryptographic services for data privacy, specifically decentralized key management and dynamic access controls.

- Value Proposition: It allows users to share private data securely with a decentralized system for managing permissions.

7. Fluence: Fluence offers decentralized data storage and processing capabilities designed for developers to build applications with privacy in mind.

- Value Proposition: It allows for the creation of decentralized applications where users can control where their data is processed and stored.

8. Storj: Storj operates a decentralized cloud storage platform, allowing users to store data in a secure, private, and distributed manner.

- Value Proposition: It separates itself by providing encrypted, distributed storage solutions, ensuring redundancy and security.

9. Celo: Initially a DeFi platform, Celo now also supports decentralized data sharing mechanisms.

- Value Proposition: Celo distinguishes itself with a mobile-first approach, aiming to expand access to financial and data services globally.

10. DIA (Decentralized Information Asset): DIA provides trustworthy oracles for DeFi applications, ensuring secure and reliable data feeds are available for blockchain networks.

- Value Proposition: It focuses on the transparency and verification of data sourced from various feeds before it's used in financial applications.

11. Aragon: While Aragon is primarily known for decentralized organizational governance, its platform can be utilized for various decentralized applications, including data marketplaces.

- Value Proposition: Offers tools for creating and managing decentralized autonomous organizations, which can be customized for various use cases including data sharing.

12. Sia: Sia is a decentralized storage platform that allows users to rent storage space in a secure and distributed network.

- Value Proposition: Focuses on low-cost solutions for cloud storage with enhanced security and redundancy.

13. Kyber Network: Primarily a decentralized exchange (DEX) protocol, Kyber also supports decentralized transactions, including data-related transactions.

- Value Proposition: It facilitates instant token exchange services on the blockchain without requiring trust in an intermediary.

14. Kleros: Kleros provides a decentralized platform for dispute resolution, which can be applied to a variety of contexts including data marketplaces.

- Value Proposition: It uses crowdsourcing and blockchain to arbitrate disputes, offering a transparent and fair mechanism.

15. Syscoin: Syscoin offers a blockchain platform that supports a variety of decentralized applications and services, including potential for data marketplaces.

- Value Proposition: Combines the benefits of Bitcoin's security with Ethereum's smart contract functionality.

These platforms, while varied in their specific focus and technology, collectively contribute to the evolving landscape of decentralized technologies. They offer Value Propositions for data sharing, trading, storage, and governance, each adding a layer of innovation and specialization in the decentralized ecosystem.

2.2 Pricing & Trading Strategies

FAME is implementing, as described in D4.2 “Pricing, Trading and Monetization Techniques” a data-driven pricing advisory mechanism leveraging issuer-provided data, stakeholder survey responses, and historical pricing realization analytics. In the next phase, the strategy intends to incorporate interest-based metrics derived from the quantification of user interactions (clicks) with analogous assets on the FDAC platform. Nonetheless, the determination of asset pricing within the FAME ecosystem remains the prerogative of the client, as explicitly articulated within the Asset Offering documentation.

2.3 Initial Business Models List

An initial list of 18 business models suitable for FAME marketplace is provided below. To ease readability, three main categories have been identified: (i) **direct revenue models**: models generating revenue directly from user transactions or services; (ii) **indirect revenue models**: models generating revenue through means not directly charged to the user; and (iii) **enhanced transaction & service frameworks**: infrastructure and services that enhance the marketplace's value proposition and operational efficiency.

Direct Revenue Models

- Subscription: Regular payments for ongoing access to data services, securing stable revenue.
- Pay-As-You-Go (PAYG): Fees based on the actual usage or transaction volume, offering flexibility.
- Pay-As-You-Use (PAYU): Fees based on the specific services or features used, offering granularity and fair pricing.
- Brokerage: Commission from facilitating data transactions between buyers and sellers.
- Razor and Blade: Low-cost core service with high-margin complementary services or data.
- Reverse Razor and Blade: High-value initial offer with revenue from subsequent lower-cost purchases.
- E-Commerce: Online platform for buying and selling data products directly.
- License Model: Fees for the rights to use data or services for a specified period.

Indirect Revenue Models

- Freemium: Basic services offered for free with premium features available for a fee, aiming for conversion.
- Hidden Revenue: Indirect revenue streams such as advertising, sponsorships, or partnerships, while offering core services at no charge.
- API Licensing: Revenue from providing third-party access to marketplace APIs.
- Fractionalization: Selling partial access or usage rights to data services for flexibility.
- Data monetization: Leveraging aggregated data to generate insights valuable to third parties (adhering to privacy laws and ethics)

Enhanced Transaction & Service Frameworks

- Peer-to-Peer (P2P): Direct transactions between users, reducing costs and fostering community.
- Data-as-a-Service (DaaS): On-demand data provision, enhancing user experience and service comprehensiveness.
- Product-as-a-Service (PaaS): Continual service provision rather than one-off product sales, improving customer engagement.
- Direct-to-Consumers (D2C): Eliminating intermediaries to enhance margins and customer relationships.
- Instant News: Real-time data sharing platform, making the marketplace a primary source for timely information.

2.4 Business Models Selection Matrix

The Business Models Selection Matrix analyses the identified business models through 5-criterias, depicted below, scoring from 1 (*low*) to 3 (*high*), to rank them thus selecting the best options.

1. **Revenue Potential**: This measures the capacity of a business model to generate income consistently over time, considering both direct and indirect revenue streams.
2. **User Engagement**: Evaluates how well a business model encourages regular interaction and long-term commitment from users.
3. **Flexibility and Scalability**: Assesses the ability of the business model to adapt to changing market conditions and user needs, as well as its capacity to scale up as the marketplace grows.
4. **Operational Complexity**: Considers the ease of implementation, management, and maintenance of the business model from the marketplace operator's perspective.
5. **Value Proposition & Fairness**: Examines how compelling the business model is to both sellers and buyers in terms of the benefits it offers, such as cost efficiency, access to valuable data, service quality, and fairness.

Table 2 - Business Models Selection Matrix

Business Model	Revenue Potential	User Engag.	Flexibility & Scalability	Operational Complexity	Value Prop. & Fairness	Total
Subscription	3	2	3	3	3	14
PAYG	3	3	3	2	3	14
PAYU	3	3	3	2	3	14
Brokerage	2	1	1	3	2	9
Razor and Blade	2	2	1	3	1	9
Reverse Razor and Blade	2	2	1	3	1	9
E-Commerce	2	2	2	2	2	10
License Model	2	1	2	3	3	11
Freemium	2	3	3	2	3	13
Hidden Revenue	2	1	2	3	2	10
API Licensing	2	2	2	2	2	10
Fractionalization	1	2	3	2	2	10
Data monetization	3	1	2	3	1	10

P2P	2	3	3	2	2	12
DaaS	2	3	3	2	3	13
PaaS	2	3	2	2	3	12
D2C	2	3	3	2	2	12
Instant News	1	3	1	3	2	10

The three selected business models are: **Subscription**, **Pay-as-you-Go (PAYG)**, and **Pay-as-you-Use (PAYU)**.

2.5 Pilots' Validation

Following Business Models selection Approach, the following step was the Pilots' Validation. A survey was shared to gather their feedback and validation, with the results below.

Table 3 - Pilots' Validation Survey

Business Model	Pilot#1	Pilot#2	Pilot#3	Pilot#4	Pilot#5	Pilot#6	Pilot#7
Subscription	x		x	x	x	x	x
PAYG	x		x		x	x	x
PAYU	x	x	x		x		x
Other			Freemium		Freemium		DaaS

FAME will therefore explore Freemium models for critical mass creation vs Subscription Based with two variants: monthly & volume-based subscription. This follows the paradigm of most financial data vendors (Bloomberg, Thomson etc.), which refrain from one off license sale to maximize the registrant adoption of the FAME marketplace. Freemium limited functionality public versions will complement paid subscription-based offerings for the use cases proprietary solutions. Moreover, DaaS will facilitate a dynamic environment where data can be accessed and monetized, and PAYG and PAYU models will introduce an element of financial flexibility and granularity, appealing to a diverse range of users with varying data needs and budget constraints. Thanks to such hybrid approach, the FAME marketplace will foster a vibrant data trading ecosystem.

2.6 Selected Hybrid Business Model & Variants

The selection of business models for FAME Marketplace namely, Subscription-based, Pay-as-You-Go, Pay-as-You-Use, Data-as-a-Service, and Freemium, has been carefully considered to align with the unique characteristics of FAME marketplace and to optimize value for both buyers and sellers.

Subscription-based Model

The Subscription-Based business model adopted by FAME Marketplace is driven by the commitment to deliver a predictable revenue stream and cultivate long-term user engagement. This model requires buyers to subscribe to a recurring payment plan, granting them access to premium features, enhanced services, and a seamless, uninterrupted experience on the platform. Subscribers benefit from the stability of a consistent pricing structure, which not only provides financial predictability for FAME Marketplace but also fosters a sense of loyalty and commitment among buyers. This model encourages buyers to invest in the platform for the long term, as they receive ongoing value and a differentiated experience, creating a mutually beneficial relationship that contributes to the sustained growth and success of our marketplace.

- Fixed-term subscriptions: Customers subscribe for a specific duration, such as monthly, quarterly, or annually (e.g. software licenses).
- Metered/Usage-based subscriptions: Customers pay based on their usage or consumption of the product or service (e.g. cloud computing or data storage).
- One-time setup fee + recurring subscription: In addition to the regular subscription fee, customers pay a one-time setup or initiation fee. This is often seen in services that require significant onboarding or adaptation.
- Corporate/Enterprise subscriptions: Tailored for larger organizations, providing additional features, scalability, and support. Usually involves custom pricing based on the size and needs of the enterprise.
- Trial subscriptions: Offers a limited-time free trial period for potential customers to experience the product before committing to a paid subscription.
- Multi-platform subscriptions: Allows users to access the subscription service across various devices or platforms (e.g. streaming services that support viewing on smartphones, tablets).

Pay-as-You-Go Model

The Pay-as-You-Go (PAYG) business model is strategically chosen to offer buyers maximum flexibility and cost control. With this model, buyers are charged based on their actual usage or transaction volume, allowing them to align their expenses directly with their business activity on the platform. The PAYG model is particularly appealing to buyers who may be exploring the marketplace or operating on a smaller scale, as it eliminates the need for a fixed, upfront commitment. This flexibility enables buyers to scale their engagement with the platform organically, paying only for the services they utilize. By adopting a PAYG approach, FAME aims to attract a diverse range of buyers, accommodate varying business sizes, and create an environment that encourages experimentation and growth without imposing rigid financial constraints.

- Usage-based pricing: Customers are charged based on the actual consumption of the product or service (e.g. in cloud computing, where customers pay for the number of server hours, data storage, or data transfer).
- Metered billing: Charges are calculated based on measured usage, often using meters or counters (e.g. a software service might charge based on the # of transactions or API calls)
- Per-transaction pricing: Customers are billed for each individual transaction or action (e.g. in financial services, where each transaction incurs a separate charge).
- Time-based billing: Customers are charged for the amount of time they use a particular service (e.g. in services like consulting or virtual office spaces).
- Pay-per-feature: Customers pay for specific features or functionalities they use. Often seen in software-as-a-service (SaaS) platforms with modular features.
- Dynamic pricing: Prices vary based on demand, time of day, or other dynamic factors. Used in industries like transportation (ride-sharing) and hospitality.
- Freemium with PAYG upgrades: Offers a basic version of the product free, with customers paying for additional usage or premium features (e.g. used in software and online platforms).

Pay-as-You-Use Model

The Pay-as-You-Use (PAYU) business model adopted by FAME Marketplace is founded on the principle of providing buyers with maximum flexibility and cost efficiency. In this model, buyers are charged based on the specific services or features they utilize, allowing for granular pricing that aligns precisely with their usage patterns. This approach ensures that buyers only pay for the resources and tools they need, promoting fairness and transparency. The PAYU model is particularly attractive to

buyers with diverse needs and budget constraints, as it eliminates the burden of fixed costs associated with subscription models. By offering this adaptable and scalable pricing structure, FAME Marketplace aims to empower buyers to optimize their expenses and tailor their engagement on the platform, ultimately fostering a thriving and dynamic marketplace ecosystem.

- Metered billing: Charges are based on measured usage, often tracked through meters or counters. (e.g., in cloud computing where customers are billed for the number of hours a virtual machine is running, data storage usage, or data transfer).
- Per-transaction pricing: Charges are incurred for each individual transaction or unit of service used. Often found in financial services, payment gateways, and API services.
- Time-based billing: Customers pay for the time they use a particular service or resource (e.g. in services like virtual office spaces).
- Pay-per-feature: Customers pay for specific features or functionalities they use (e.g. in SaaS models where users pay for additional features beyond the basic package).
- Dynamic pricing: Prices fluctuate based on dynamic factors such as demand, time of day, or other contextual variables.
- Freemium with usage-based upgrades: A combination of a free basic version and the option to pay for additional usage or premium features (e.g. in software and online platforms).

Data as a Service (DaaS) plays a pivotal role in FAME marketplace by ensuring the availability of diverse data sources, ranging from demographic information to transactional data and market research. DaaS fosters interoperability among these varied data sets and systems, allowing users to seamlessly access and integrate data from multiple providers. Beyond facilitating data accessibility, DaaS presents monetization opportunities for data providers, who can capitalize on their data assets by offering them as services within the marketplace. This not only generates revenue streams for data owners but also furnishes users with invaluable insights and information, enriching their experience within the marketplace.

- Raw Data Access: Pilots provide direct access to raw datasets through APIs or data feeds. Users can query, retrieve, and manipulate the data according to their specific needs. Pricing may be based on factors such as data volume, API calls, or data refresh frequency.
- Data Aggregation: DaaS providers aggregate data from multiple sources into a unified dataset, making it easier for users to access and analyze diverse data sources in one place. Pricing may be based on the number of sources aggregated or the complexity of the data integration process.
- Subscription-based Access: DaaS providers offer access to data through subscription-based pricing models, where users pay a recurring fee for ongoing access to data services. Subscription plans may vary based on factors such as data volume, access frequency, or the level of service provided.
- Pay-per-Use: Users are charged based on the actual usage of data services, such as the number of API calls made, the volume of data processed, or the duration of data access. This model offers flexibility for users who may have fluctuating data needs but requires careful monitoring to avoid unexpected costs.
- Freemium Data Access: Similar to freemium models in other industries, DaaS providers offer basic access to data for free, with premium features or additional data available for a fee. This allows users to explore the data and evaluate its usefulness before committing to a paid plan.

The **Freemium model** within FAME marketplace serves as a dynamic strategy for user acquisition, offering basic access to data for free to attract a broad user base and stimulate adoption among potential customers. These free tiers not only entice users to explore the platform's capabilities but

also serve as a pathway for upselling opportunities. Once users recognize the value in the provided data and become accustomed to its basic features, they are more inclined to upgrade to premium plans or purchase additional data services. This not only enhances user engagement but also drives marketplace growth by expanding the ecosystem. As more users join the platform and interact with the available data, the marketplace gains momentum and becomes increasingly valuable for both data providers and consumers alike.

- Feature-based freemium: This model offers a basic version of the product with limited features for free, while premium features are unlocked with a paid subscription. Users can choose to upgrade based on the specific features they need.
- Time-limited freemium: In this model, users have access to the full functionality of the product for a limited time, such as a trial period. After the trial period expires, users are required to pay for continued access.
- Capacity-based freemium: Here, users can access the product up to a certain capacity or usage limit for free. Once they reach the limit, they are prompted to upgrade to a paid plan to continue using the product without restrictions.
- User-based freemium: This model limits the number of users who can access the product for free. Businesses or teams with more users than the free tier allows must upgrade to a paid plan to add additional users.
- Transactional freemium: This model offers a basic version of the product for free but charges for individual transactions or premium upgrades within the product. Users can continue using the basic version for free but pay for specific actions or enhancements.
- Hybrid freemium: Combining elements of various freemium models, the hybrid approach may offer different tiers with combinations of feature limitations, time limits, or usage thresholds to cater to different user needs.

By incorporating a combination of these business models, FAME Marketplace seeks to strike a balance between revenue predictability, user engagement, flexibility, and fair pricing. This strategic alignment is aimed at creating a marketplace ecosystem that not only accommodates the diverse needs of our users but also encourages sustained growth and participation from a wide array of sellers and buyers. As market dynamics evolve, the ability to offer multiple business models positions FAME Marketplace to adapt swiftly and stay competitive in the ever-changing landscape of online marketplaces.

3 Module Overview

The Operational Governance (GOV) module of the FAME Platform is, as its name implies, the entry point for all management and governance functions needed for operating an instance of the FAME federation. There are five types of *information entity* that are managed by the GOV module:

- Federation Member – Members (organizations) of the FAME federation.
- User – Users of the FAME platform. Note: typically, users of the FAME platform are onboarded and managed externally, by Federation Members having the Onboarding Authority role; however, the platform has the capability of managing its own *internal* users, that are granted special privileges like ADM ("administrator") and SUP ("user support") by a special kind of entity, classified as Root Authority.
- Support Ticket – Requests for assistance filed by users, to be processes in the background (asynchronously) by a human operator.
- User Request – Requests submitted by client software, to be processed in the background (asynchronously) by a service endpoint.
- Trading Account – Blockchain accounts that are used for trading on the FAME marketplace.

Each of these entity types is supported by a dedicated *database*, where the relevant information is maintained the GOV module, which provides several *operations* – i.e., service calls defined by the GOV API – that users can execute to store, retrieve, or modify information. Some operations also have *side effects*, meaning that they can reach beyond GOV-managed information and impact on other modules; for instance, enrolling a Trading Account (see the operation list below) implies *granting transaction permission* to that account on the FAME Blockchain Infrastructure (which is done by the Provenance & Tracing module – see deliverable D4.1).

All operations follow the REST[1] approach, and thus can be executed in any order at any time. The authorization rules are operation-specific, but always match one of three options: admin users only, any authenticated users, anyone.

At the time of writing, the GOV module is designed to implement the operations listed below. As FAME adopts an agile approach to design and development, *these may change in the next release of the software*.

- Operations related to Federation Members:
  - **Onboard Member** – Registers a new Federation Member, creating a record with a system-assigned ID and the information provided by the caller.
  - **Update Member** – Updates a Federation Member record with the given data.
  - **Offboard Member** – Disables a Federation Member, reversing the onboarding operation but keeping the existing registration record.
  - **Delete Member** – Permanently removes an offboarded Federation Member from the system.
  - **List Members** – Retrieves the list of Federation Members matching the given criteria.
  - **Retrieve Member** – Retrieves all the information available for a given Federation Member.
  - **Lookup Authority** – Checks if a given DID corresponds to a currently onboarded Federation Member.
- Operations related to Users:

[1] See https://www.seobility.net/en/wiki/REST_API

  - **Invite User** – Registers a new candidate user with the information provided by the caller and sends an invitation message to the specified contact point(s), thus starting the onboarding process.
  - **Accept Invitation** – Allows a candidate user to accept a previously-issued invitation and to receive a one-time-password (OTP) for starting the Verifiable Onboarding Credential (VOC) issuance sequence (see Claim Credential operation).
  - **Claim Credential** – Starts the Verifiable Onboarding Credential (VOC) issuance sequence for a previously-invited candidate user who is in possession of the correct OTP (see Accept Invitation operation), thus completing the onboarding process.
  - **Update User** – Updates a User record with the given data.
  - **Renew Onboarding** – Allows an administrator to reset a (possibly stuck) onboarding process or to start a new onboarding process for an existing (possibly offboarded) user, by issuing a new invitation.
  - **Offboard User** – Revokes the onboarding of a user.
  - **Delete User** – Permanently removes a User record from the system.
  - **List User** – Retrieves the list of User records matching the given criteria.
  - **Retrieve User** – Retrieves all the information available for a given user.
- Operations related to Support Tickets:
  - **Open Ticket** – Creates a new Support Ticket in the system, with a system-assigned ID, status set to “pending” and the caller's UID as "owner".
  - **Receive Ticket** – Sets the status of a "pending", "processing" or "suspended" Support Ticket to “processing” and sets the caller's UID as "assignee".
  - **Reply to Ticket** – Appends an entry to the history of a Support Ticket with "processing" status.
  - **Suspend Ticket** – Sets the status of a "processing" Support Ticket to “suspended”; additionally, if a public and/or internal message are provided by the caller, they are appended to the history of the ticket.
  - **Close Ticket** – Sets the status of a "pending" or "processing" Support Ticket to “closed”, and its "outcome" to the given value; additionally, if a public and/or internal message are provided by the caller, they are appended to the history of the ticket.
  - **Reopen Ticket** – Sets the status of a "closed" Support Ticket to “processing” and sets the caller's UID as "assignee"; additionally, if a public and/or internal message are provided by the caller, they are appended to the history of the ticket.
  - **Update Ticket** – Updates a Support Ticket with the given data, regardless of the workflow state.
  - **List Tickets** – Retrieves the list of Support Tickets matching the given criteria.
  - **Retrieve Ticket** – Retrieves all the information available for a given Support Ticket.
- Operations related to User Requests:
  - **Check Request status** – Verifies the status of a previously submitted User Request.
  - **Enqueue Request** – Creates a new "pending" User Request in the system.
  - **Retrieve Request** – Retrieves the given User Request record.
  - **Finalize Request** – Updates a User Request record with the given data and status.
- Operations related to Trading Accounts:
  - **Enroll Account** – Registers a Blockchain account as a Trading Account owned by the caller.
  - **Disenroll Account** – Disables a given Trading Account, reversing the enrollment operation.

- **Credit Account** – Mints a given amount of FDE coins and transfers them to a given Trading Account.
- **Debit Account** – Burns a given amount of FDE coins from a given Trading Account.
- **List Accounts** – Retrieves the list of Trading Accounts matching the given criteria.
- **Retrieve Account** – Retrieves all the information available for a given Trading Account.
- **List Transactions** – Retrieves the list of transactions related to the transfer of FDE coins affecting a given Trading Account, matching the given criteria.
- **Retrieve Balance** – Retrieves the amount of FDE coins currently owned by given Trading Account.
- **List Traders** – Retrieves the list of users that own one or more Trading Accounts and match the given criteria.
- **Retrieve Trader** – Retrieves all the information available for a given user that owns one or more Trading Accounts.
- **List Affiliations** – Retrieves the list of Trusted Sources linked to Trading Accounts.
- **Retrieve Affiliation** – Retrieves all the information available for a given Trusted Source linked to Trading Accounts.
- **Lookup Ownership** – Given the identifier of a currently enrolled Trading Account, returns the linked user and affiliation.
- **Retrieve Balance for Token** – Retrieves the number of tokens of a given type that are currently owned by given Trading Account.
- **Mint / Burn Tokens** – Mints a given amount of a given token and transfers them to a given Trading Account OR burns a given amount of a given token from a given Trading Account.

Additionally, the GOV module implements all the service endpoints defined by the OpenID for Verifiable Credential Issuance (OID4VCI) specification[2] – namely, the Credential Offer, Authorization and Token endpoints. This is required in order to deliver the required Verifiable Onboarding Credentials (VOC) when onboarding Users.

From a software architecture point of view, all the above operations are implemented by four distinct components: **Federation Management** (deals with Federation Members), **User Management** (deals with Users and includes the OID4VCI implementation for issuing VOCs), **User Support** (deals with Support Tickets and User Requests), and **Trading Support** (deals with Trading Accounts). Individual operations will be thoroughly documented in §4.2, §4.3, §4.4, and §4.5, respectively.

At the time of writing (May 2024), and in the context of the present deliverable, the design and technical specification of GOV module are nearly complete, with minor adjustments foreseen before its final release (expected by M28). On the other hand, its concrete implementation is only partial: the Module Demonstration section (see §5) will provide the details on which parts are included in the current prototype release.

3.1 Positioning into FAME SA

In the *C4 context-level diagram* of the FAME Solution Architecture, the GOV module is classified as a *container* named Operational Governance within the Transactional Operations *system* – see Figure 2. The cooperation relationships identified there are with the Authentication and Authorization module and with the Assets Trading & Monetization module. The is also a mention of Service Level Agreements (SLA) as a GOV internal concern. It is worth noting, however, that the diagram depicted

[2] See https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html

here comes from deliverable D2.3, which was released at an earlier stage of the project. Since then, due to the agile nature of our development process, cross-module relationships did undergo some revisions: for an up-to-date version of these, please refer to the *C4 container-level diagram* of the GOV module provided in §3.2 as Figure 3. Additionally, **SLA management has been moved under the scope of task T4.1**, as the definition of a proposed SLA – if any is provided by an asset publisher – is now done as part of the *offering definition* (see deliverable D4.1), while its enforcement is not in the scope of the FAME Platform at all.

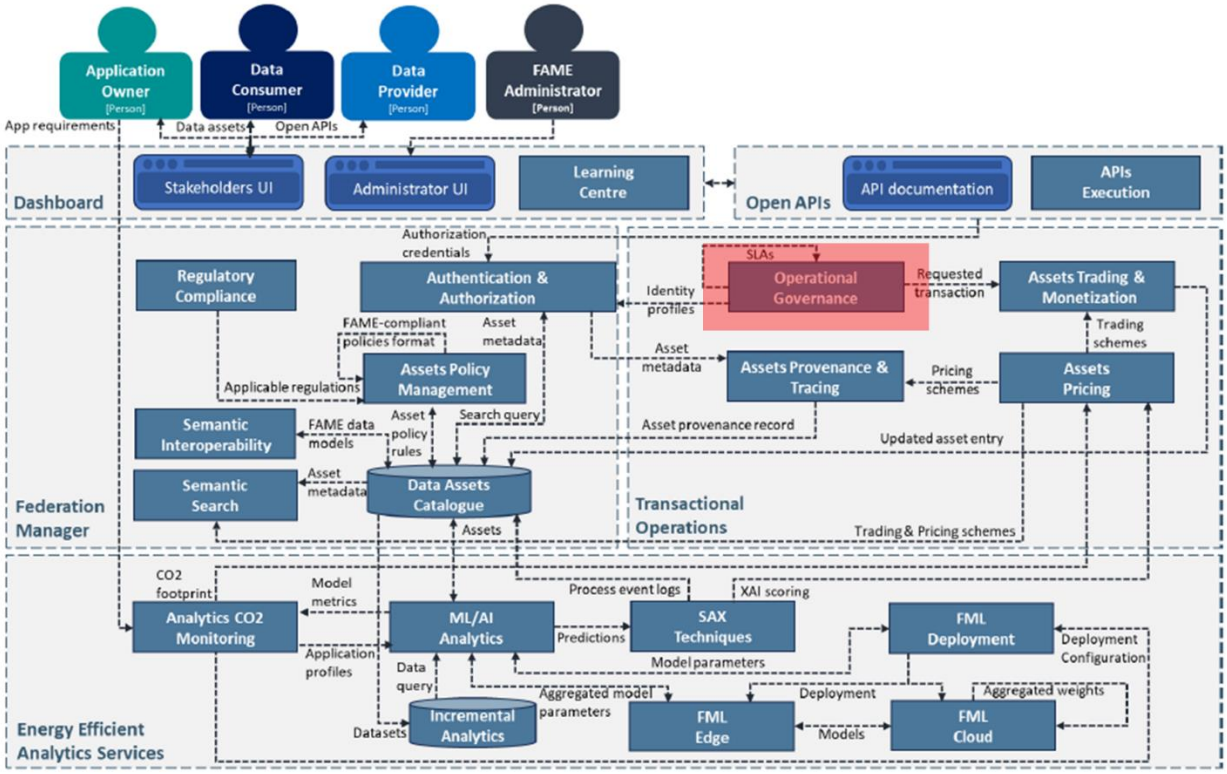

Figure 2 – FAME Solution Architecture: C4 context-level diagram (GOV container highlighted)

3.2 C4 Component-level Architecture

The goal of the *Operational Governance* platform module is to provide the necessary services for the management and the day-to-day operation of the platform. These services are exposed by four components: Federation Management, User Management, User Support, and Trading Support.

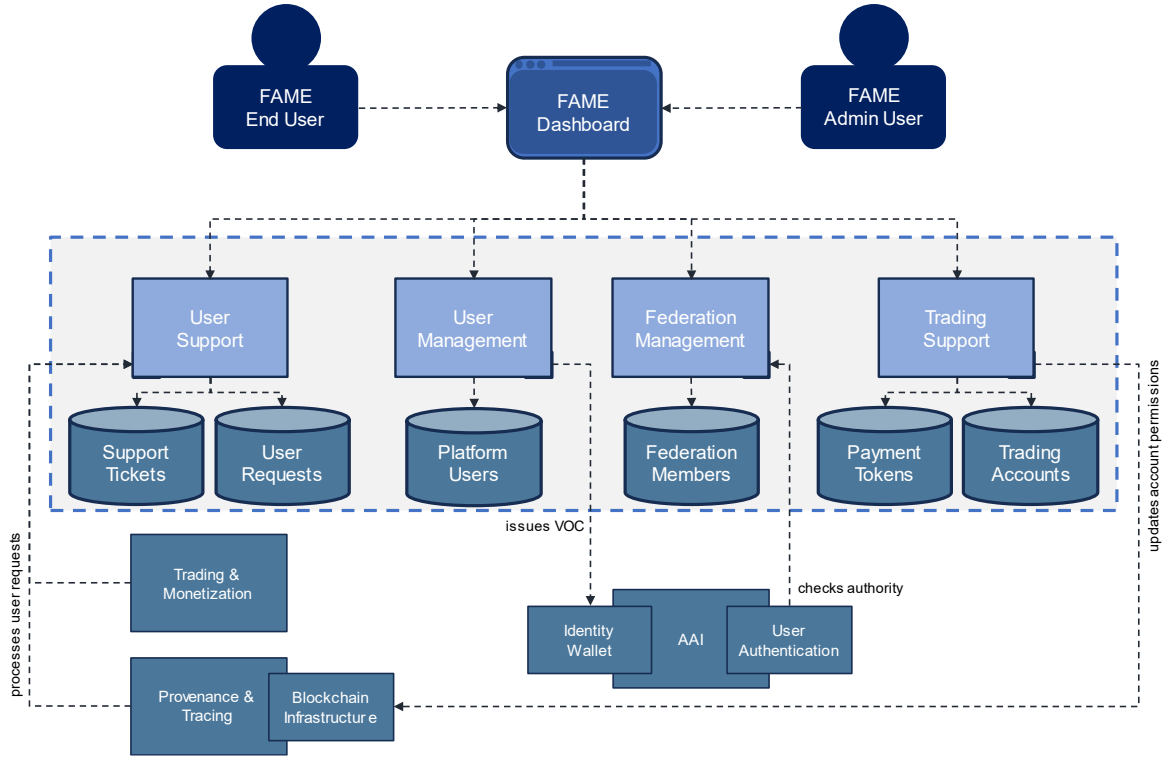

Figure 3 - GOV Container: C4 Component-level diagram

3.3 Workflows and Integration with Other Modules

The GOV workflows that are worth mentioning are those that enable the following objectives:

- Onboarding of Federation Member
- Onboarding of User
- Enrollment of Trading Account
- Procurement of FDE coins

For each of these, GOV also supports a workflow that operates in the opposite direction - e.g., "Offboarding of Federation Member", "Redemption of FDE coins" - but we are not going to describe them here, for the sake of brevity.

In the step-by-step description of workflows that follow, some steps are performed with tools that are external to the FAME Platform. Good examples are the interactions of users with banking services (as in §3.3.4) and the onboarding of users made by external Onboarding Authorities (see §3.3.2.1). These steps can be easily spotted as *their descriptive text is in italics*.

The workflows are described assuming that all actors will interact with the GOV module through the FAME Dashboard, and that user authentication will be performed at the Dashboard level by means of the FAME Authentication & Authorization Infrastructure (AAI). With respect to the latter topic, GOV's Open API service endpoints work in the same way as all public services of the FAME platform: the security context, which has the form of a FAME-issued *token* and ensures that the caller's attributes are trustworthy, is established by the login process *in the web browser of the user*; it is then retrieved by the Dashboard web page and finally propagated to the Open API service endpoint. Access control decision (i.e., authorization and/or content filtering) are made, case by case, by the web page and *repeated* by the service endpoint (to ensure that direct calls will not bypass any control). Although the details of this process are not relevant for this deliverable, a high-level picture of it is provided here for better understanding.

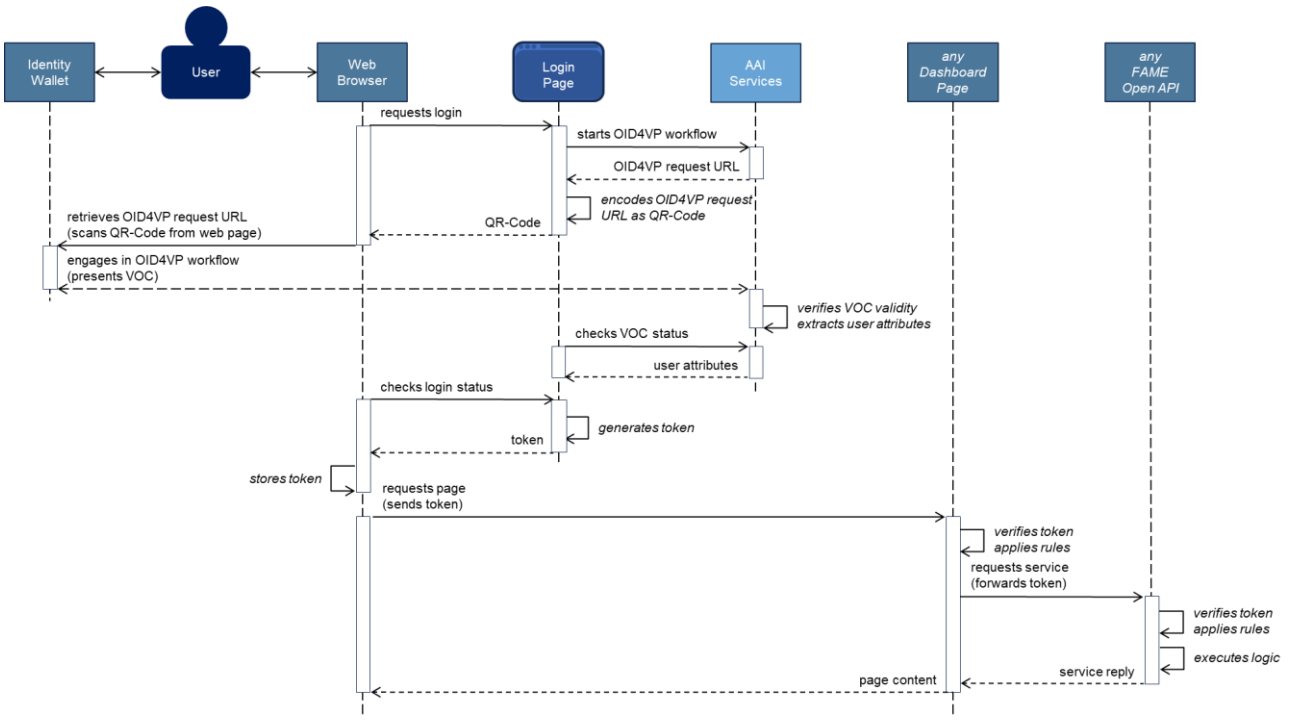

Figure 4 - Access control workflow (simplified diagram)

3.3.1 Onboarding of Federation Member

This process happens mostly offline, the only involvement of the FAME Platform being the last step. From the FAME Platform's perspective, its purpose is to add a Federation Member information entity to the relevant registry, so that VOCs issued by the new member are accepted by FAME's AAI.

1. *The candidate organization goes through an offline process to have its membership approved by the FAME federation governance board (FGB). As part of this process, its legal identity and contact information are communicated to the FGB and verified. If the new member must operate as an Onboarding Authority, the process will also require the member to set up a suitable VOC issuance system[3], and the FGB to assess it compatibility with FAME's specifications.*
2. *The admin user receives instructions from the FGB to proceed with the onboarding of the new member. The instructions include the new member's contact information. If the member must operate as an Onboarding Authority, the admin user will obtain from it the OA's Decentralized Identifier (DID).*
3. The admin user connects to the FAME Admin Dashboard, going through the authentication & authorization process.
4. The admin user navigates to the Members section and registers the newly approved member as a Federation Member entry in the platform's registry.

[3] The User Management component (see §4.3) provides a simple implementation of such system, which is only used by the FAME Root Authority to manage admin users. Typically, external Onboarding Authorities will need a much more complex system, integrated with their internal user management software.

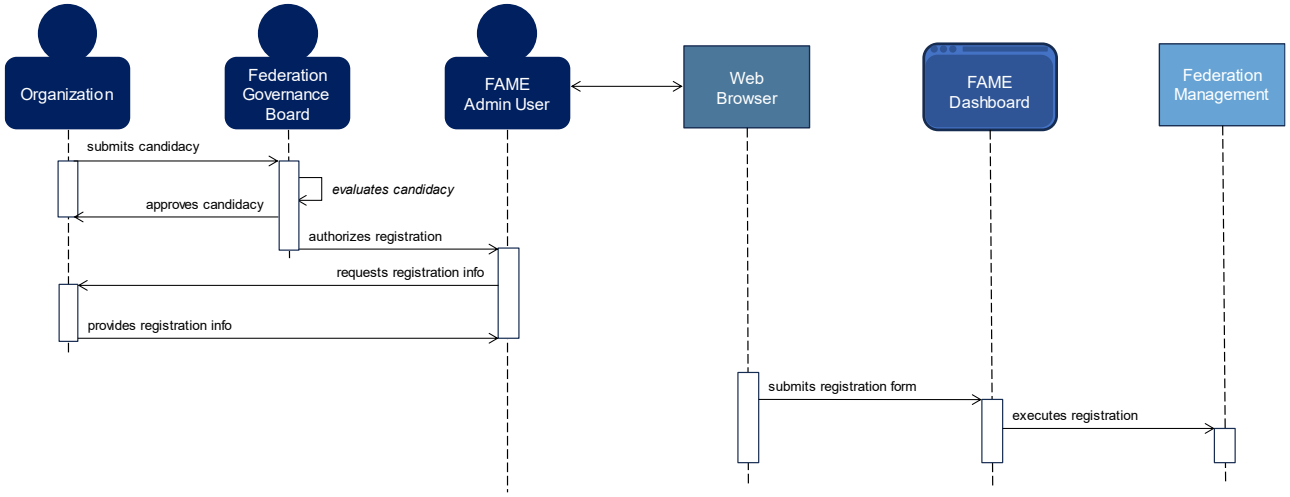

Figure 5 - Onboarding of Federation Member workflow

3.3.2 Onboarding of User

There are two similar but different processes for the onboarding of users:

- **Regular users** are people who participate in the FAME data marketplace as traders or guests: they are onboarded by an Onboarding Authority (OA), which is a legal entity that is member of the FAME federation.
- **Platform operators** are people having administrative tasks on the FAME Platform: they are onboarded by a Root Authority (RA), which is an organization that is *a core member* of the FAME federation (typically, this means that it directly contributes to hosting/running the IT infrastructure that underpins the FAME Platform).

Regardless of being an OA or RA, such role implies some responsibilities:

- Verification of user's identity.
- Management of any personal data the authority is in possession of.
- Compliance with all regulation that applies to such personal data (e.g., GDPR).

To avoid placing the big burden of GDPR compliance - and associated risks and costs - on the core members of the FAME federation, the onboarding and management of the *vast majority* of users is delegated to the OAs. Moreover, FAME's architecture is designed so that no personally identifiable information (PII)[4] is transferred from the OA to the platform. On the other hand, a RA is only responsible for a handful of users: platform operators, *who will typically already have employee status*.

From the GOV module's perspective, these two scenarios are very different. The first is not supported at all, because all onboarding activities are performed on the OA's side, resulting into a VOC that carries no PII and that will only be checked for validity when presented by the user (in other words: the process is a concern of the AAI module). The second is supported in the simplest of ways: platform operators are registered with a minimum of contact information, and VOC issuance is a straightforward online workflow that requires users to follow an invitation link to a web page and scan a QR-Code with their FAME Identity Wallet app.

[4] See https://en.wikipedia.org/wiki/Personal_data

3.3.2.1 Onboarding of Regular User

1. *The candidate is known to the OA[5] and goes through an internal approval process. If no identity verification was done previously, it must be included in the process[6].*
2. *The candidate, now an approved but not yet onboarded FAME user, obtains the FAME Identity Wallet app - or any other OID4VC-compatible app - and installs it on their personal mobile device.*
3. *The user, through any OA-specific process, retrieves their VOC and stores it in their wallet app.*

3.3.2.2 Onboarding of Platform Operator

1. *The candidate, who is an endorsed but not yet onboarded FAME platform operator, obtains the FAME Identity Wallet app and installs it on their personal mobile device.*
2. The admin user connects to the FAME Admin Dashboard, going through the authentication & authorization process.
3. The admin user navigates to the Operators section and registers a new User entity, inserting the required contact information and setting the assigned role[7].
4. The system sends an invitation message to the email address - and, if provided, to the phone number - included in User entity. The message contains a link to a dedicated Dashboard page, together with an *invitation ID* that is specific to the current onboarding workflow.
5. The invited user follows the received link, thus opening the Dashboard page in their web browser. The page prompts the user to submit a one-time-password (OTP) in order to proceed further.
6. The system, having detected the user and checked the status of the pending invitation, generates a workflow-specific OTP and sends it to the user's contact point(s) - see step #3.
7. The invited user submits the received OTP through the Dashboard page.
8. The system verifies that the OTP is correct and issues the user's VOC. It then displays a QR-Code that encodes the information needed for the retrieval of the VOC from the platform.
9. The invited user activates their wallet app and scans the QR-Code.
10. The wallet app, after asking confirmation from the user, retrieves the VOC from the platform and stores it locally.

[5] The candidate may be an OA's employee, but also a registered user of some online service operated by the OA - e.g., a data space.
[6] Although the approval process is not defined by the FAME architecture, the OA is taking on responsibility for future actions of the user in the FAME ecosystem. Moreover, the confidence level the OA has in the correct identification of the user must be high enough to mitigate the risk of failing to comply with existing regulation.
[7] Currently, the supported roles are platform administrator and user support operator.

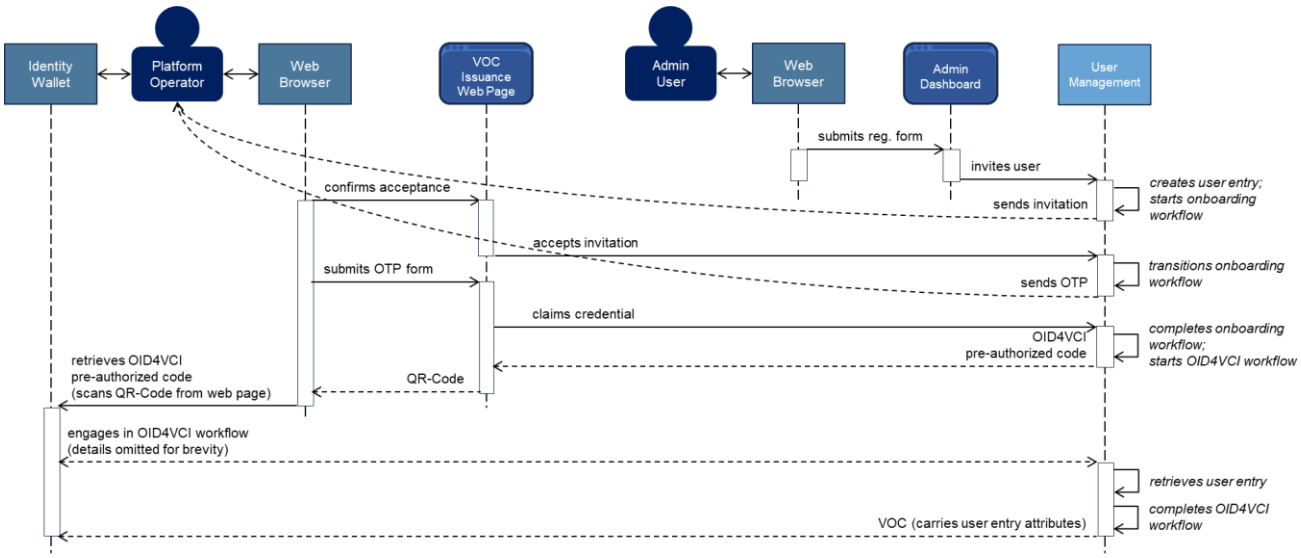

Figure 6 - Onboarding of Platform Operator workflow

3.3.3 Enrollment of Trading Account

Trading Accounts are a staple of FAME's data marketplace: they ensure that all the business models supported in FAME (see §2.6) can work by means of *online transactions*, where access to data assets is unlocked by a real-time payment[8]. A Trading Account is a FAME *extension* to a plain blockchain account. It is based on a regular, Ethereum-style account but has two additional features:

- **Permissioning** - The FAME Blockchain Infrastructure is a *permissioned* network, which means that blockchain accounts must be included in a system-managed *allowlist* to be authorized to execute any smart contract, or to submit blockchain transactions in general.
- **Pseudonymous ownership** - FAME blockchain accounts are not entirely anonymous as in the Ethereum world: they are mapped to the ID of the user who owns them - basically, to a pseudonym that, in case of need, can be traced back to a real identity with the cooperation of the OA who originally onboarded the user (see §3.3.2.1)[9].

Both features are enabled by one single workflow: Trading Account enrollment. This step is mandatory for any user who, having already gone through onboarding, wants to either buy or sell assets on the FAME data marketplace.

1. *The user, by means of any suitable blockchain wallet software, creates an Ethereum-style blockchain account and copies down the resulting Ethereum address.*
2. The user connects to the FAME End-user Dashboard, going through the authentication & authorization process. This process ensures that the incoming user is associated with the correct user ID (UID) and, if present, with the provenance ID (PID) that indicates the user's affiliation.
3. The user navigates to the Profile section, where a web form is displayed that prompts the user to insert a Trading Account ID (TID) for enrollment.

[8] Technically speaking, this is done by exchanging, on the FAME Blockchain Infrastructure, generic *payment tokens* (a digital currency) for an asset-specific *access token* (a digital proof of access rights)

[9] The FAME architecture protects the anonymity if its users and, at the same time, shields its core members from PII-related responsibilities by offloading the latter on the external Onboarding Authorities. The basic principle is that the core members don't know who the platform users are, as their only clue is the UID that is assigned to them individually by their OA. However, the FAME platform *does* track the activities originating from an UID, and the blockchain account mapping is an example of this. The scenario in which the real identity associated to a UID is revealed by the OA, however, is limited to a judiciary or law-enforcement action.

4. The user inserts the blockchain account address as the TID value and submits the form. The user's UID is also included in the submission.
5. The system checks that the given TID is valid and that is not already enrolled as a Trading Account. It then turns the enrollment request into a User Request message, which is enqueued for being processed in the background. The request ID is immediately returned to the Dashboard page, where it is used as a handle for monitoring the progress of the process.
6. While the user is waiting for status updates, the system submits - using a system-owned blockchain account - a smart contract transaction that adds the TID to the Blockchain Infrastructure's allowlist[10]. Being a blockchain transaction, this operation is asynchronous: the system must wait for a blockchain event to receive confirmation that the transaction was successful.
7. When the blockchain transaction is confirmed, the system updates the Trading Account database, adding a mapping between the TID and the UID/PID of the requesting user. It also updates the User Request message in the request queue, marking the process as successfully completed.
8. The user, still on the same page from which the enrollment request was filed, checks that their request was successfully processed.

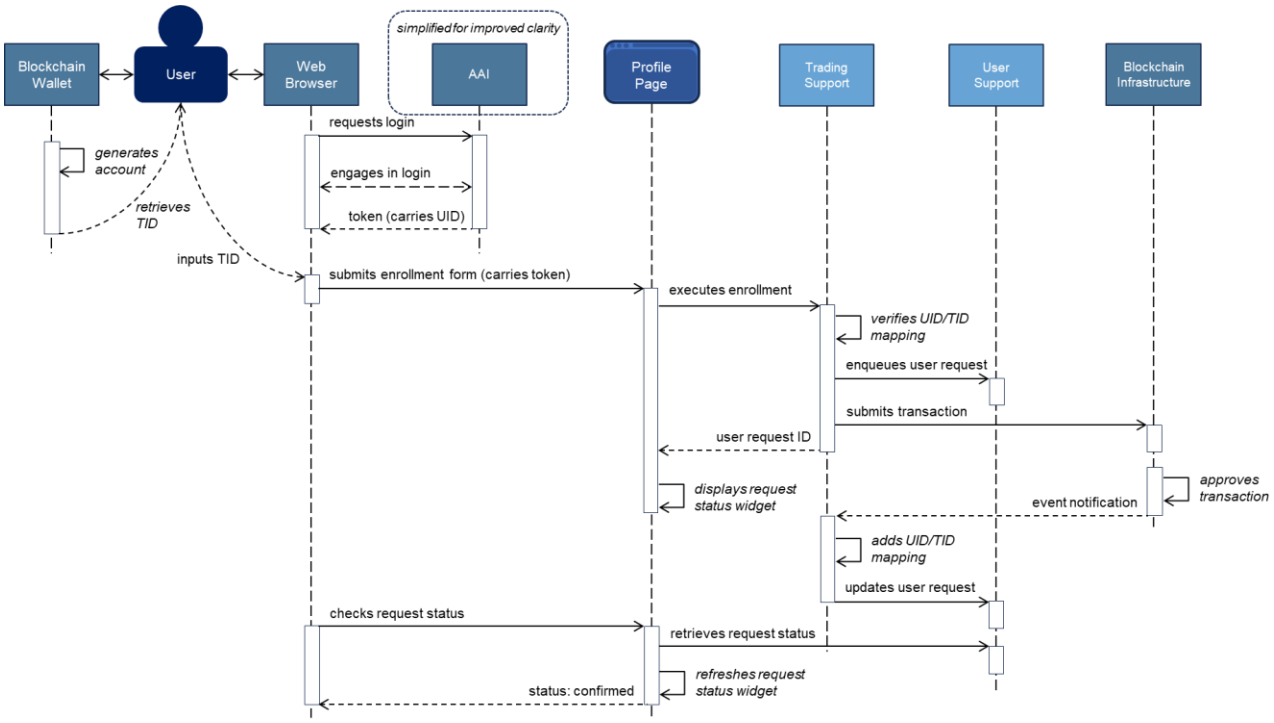

Figure 7 - Enrollment of Trading Account workflow

3.3.4 Procurement of FDE coins

The main function of Trading Accounts is to send and receive payment tokens, that in FAME are implemented as a digital currency named "FDE coin" (see §4.5.1). As FDE coins are a digital representation of the EUR currency (a pattern known as *stablecoin*), and the FAME ecosystem is not supposed to create actual money out of thin air[11], the existence of every single FDE coin is connected

[10] In the FAME Blockchain Infrastructure, the allowlist is managed *on-chain*, being a common "point of truth" for all the nodes of the network.

[11] However, during testing and demonstration of the FAME Platform - i.e., for the entire duration of the FAME project - this rule is *not* applied: FME coins are indeed managed as a valueless currency, and no FME-to-EUR or EUR-to-FDE conversion is possible.

with that of a matching EUR deposited at an *escrow account*[12]. In practice: users get FDE coins in their Trading Account - and can thus make payments in the FAME data marketplace - on condition of having sent the equivalent amount of EUR to a special, FAME-managed bank account; they also get EURs in *their* bank account on condition of having the matching amount of FDEs removed (and *burnt* by the system for good) from their Trading Account. The former workflow is called **procurement**, the latter, **redemption**.

The procurement workflow is unraveled below. The prerequisite for both is that the FAME governance board has set up the required banking agreements, remote banking software and user accounts, so that one or more banking accounts can be managed online, by one or more trusted operators, as FAME escrow accounts.

1. The user connects to the FAME End-user Dashboard, going through the authentication & authorization process.
2. The user navigates to the Profile section, and from there to the page that displays the details and status of a Trading Account owned by the user.
3. The user starts the "request funds" process, to the effect that the page displays a web form for submitting a request for receiving funds on the currently displayed Trading Account.
4. The user inputs the amount of requested FDE coins and submits the web form.
5. The system turns the request into a Support Ticket, which is created in "pending" state and contains the details of the funding request[13].
6. The operator connects to the FAME Admin Dashboard, going through the authentication & authorization process.
7. The operator navigates to the Helpdesk page, where the pending Support Ticket is listed, and takes on responsibility for it (i.e., the operator becomes the ticket's *assignee*).

NOTE: From now on, all communications between the user and the operator happen through the channel represented by the Support Ticket abstraction.

8. The user support operator sends the escrow account coordinates to the user.
9. *The user transfers EUR-denominated funds to the escrow account. The transferred amount matches the funding request.*
10. *The operator verifies that the escrow account did receive the correct amount of EUR-denominated funds.*
11. The operator navigates to the Traders section of the Admin Dashboard, and from there to the page that displays the details and status of the Trading Account for which the funding request was submitted.
12. The operator starts the "manage tokens" process, to the effect that the page displays a web form for minting and crediting FDE tokens to the currently displayed Trading Account.
13. The operator inputs the amount of FDE coins to mint and credit, and submits the web form.
14. The system submits - using a system-owned blockchain account - a smart contract transaction that executes the required steps[14].
15. In due time, the operator verifies, on the Trading Account detail page (see step #11), that the balance has the expected value. The operator then communicates to the user the completion of their support request and closes the Support Ticket.

[12] See https://www.investopedia.com/terms/e/escrow.asp

[13] From now on, the user will be able to follow the progress of the support request from the Helpdesk page of the Dashboard, where all Support Tickets are tracked. This is however out of the scope of this workflow, so it will not be mentioned any further.

[14] This operation, under normal circumstances, has no chance to fail, as it does not depend on the correctness of the input: for this reason, the system does not wait for confirmation.

16. The user, at any time, can verify that the FDE funds have been received by checking the Trading Account balance from its detail page (see step #2).

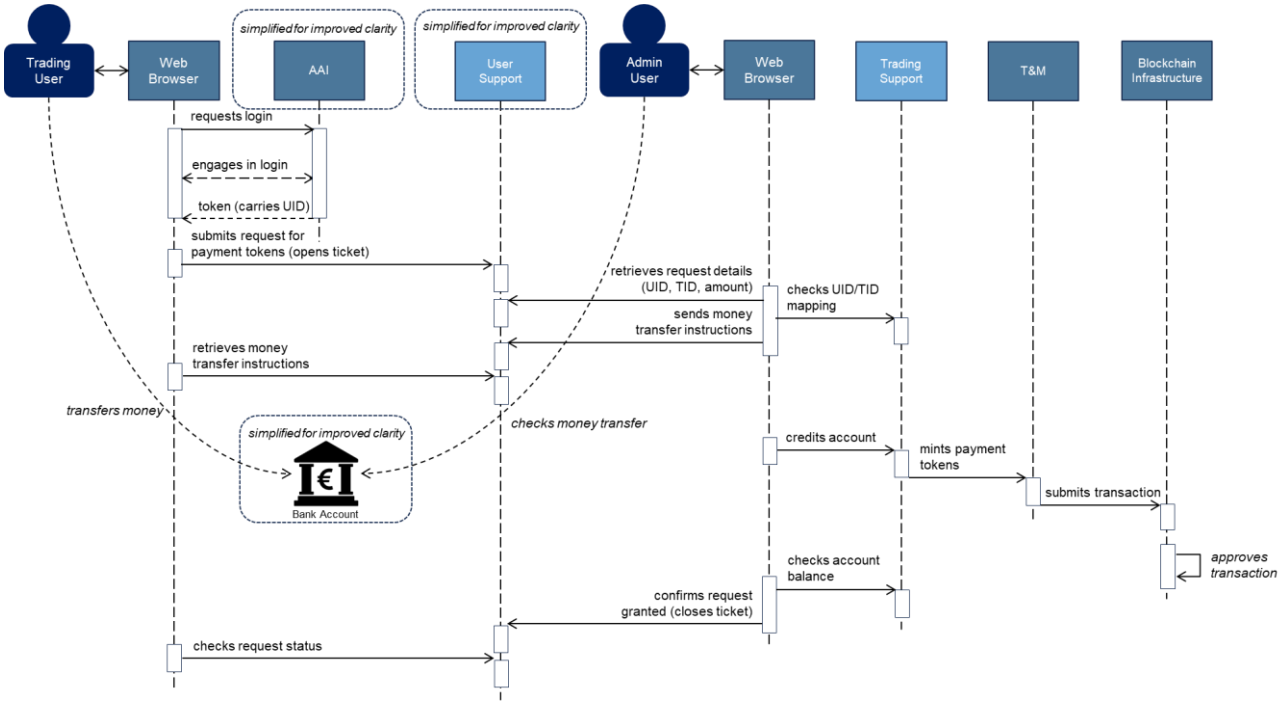

Figure 8 - Procurement of FDE Coins workflow

4 Components Specification

This section contains the technical documentation of the Open API (user- and application-level service endpoints) and Integration API (cross-module integration points) of the GOV module, grouped by component.

Open API and Integration API operations can be distinguished by looking at the "API Visibility" paragraph of their documentation entry:

- Open API operations have PUBLIC visibility: they *must* be exposed over the public Internet using the secure version of the HTTP protocol, and their access *must* be mediated by the FAME Authentication and Authorization Infrastructure. To this goal, their visibility paragraph also describes the Access Control rules that are applied - e.g., "authenticated user with ADM role".
- Integration API operations have INTERNAL visibility: they *must not* be exposed over the public Internet. It is assumed that the LAN used to connect the different modules of the FAME Platform is configured in such way that access to Integration API endpoints is only possible from local IP addresses.

All the API operations, regardless of component or visibility, follow the REST-over-HTTP paradigm[15].

4.1 Baseline Technologies and Tools

The baseline technologies and tools are listed in the table below:

Table 4 – Baseline technologies and tools

Baseline Technology	Description	Usage in FAME
Node.js https://nodejs.org/en	Runtime environment for server-side applications	Implementation of service endpoints
NestJS https://nestjs.com/	Framework for server-side applications	Implementation of service endpoints
Typescript language https://www.typescriptlang.org/	Javascript-style programming with object/data types	Implementation of service endpoints
MongoDB https://www.mongodb.com/	No-SQL database platform	Persistence for User Request entities
MySql https://www.mysql.com/	SQL database platform	Persistence for all other entities
Hyperledger Besu https://www.hyperledger.org/projects/besu	Ethereum-compatible blockchain platform	Deployment of smart contracts
Solidity language https://soliditylang.org/	Development of Ethereum-compatible smart contracts	Implementation of smart contracts
Sphereon SSI SDK https://sphereon.com/sphereon-ssi-sdk/	Framework for self-sovereign identity applications	Implementation of OID4VCI protocol

[15] See https://medium.com/@sp00m/rest-over-http-or-why-your-http-api-isnt-restful-94fd92a0e6d4

4.2 Federation Management

4.2.1 Description

The Federation Management component provides the Open API for the administration of federation members: it supports their onboarding / offboarding, the management of their records, and implements a directory for quick membership checks. The definition of "federation member", in the FAME context, is the following: a *legal entity* that has been approved by the federation's governance body as a trusted contributor to the FAME's catalogue and/or as a trusted authority for the onboarding of FAME users. Authorities fall into two categories: Onboarding Authorities (OA) are entities that are entrusted with the onboarding[16] of "regular" users - i.e., publishers and consumers of data assets; Root Authorities (RA) are a special case of OA, as they can onboard users with administrative privileges[17].

The component is, at its core, a simple and straightforward registry with the usual CRUD[18] capabilities. There is however a unique feature that stems from the fact that every member record can also include the Decentralized Identifier (DID) used by that member for issuing Verifiable Onboarding Credentials (VOC) to users when the member acts as an OA or RA. While the user onboarding process will be explained in detail in the coming deliverable D2.2 "Technical Specifications and Platform Architecture" v2 (to be released at M18), suffice to say here that the Federation Management enables the AAI layer to check that a VOC presented by a user as a *proof of onboarding* was issued by an entrusted authority.

4.2.2 Technical Specification

4.2.2.1 Interfaces

- **Onboard Member** – Registers a new Federation Member, creating a record with a system-assigned ID and the information provided by the caller.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM role.

```
POST https://<public_address>/api/v1.0/members
Request body (Content-Type: application/json)
{
  "typ": "..",          // member type (MTYPE domain)
  "did": "..",          // decentralized id (optional if type is OTHER)
  "cty": "..",          // country (ISO 3166-1 alpha-2)
  "name": "..",         // organization name
  "desc": "..",         // organization description (optional)
  "email": ".."         // contact point (optional)
}
Response body (Content-Type: application/json)
{
  "mid": ".."           // member id (assigned by system)
}
```

- **Update Member** – Updates a Federation Member record with the given data.
  Note: all attributes provided by the caller are optional, missing ones will not be changed.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM role.

```
PUT https://<public_address>/api/v1.0/members/{mid}
Path parameters:
```

[16] Onboarding requires the verification of the user's real identity as a physical person, the assignment of a user profile that determines how the user is allowed to interact with FAME services, and the issuance of a Verifiable Onboarding Credential - see later in the chapter.
[17] Specifically, users having `ADM` or `SUP` role - see the `UROLE` domain in §□.
[18] Create, Read, Update, Delete operations on database records.

```
• mid - member id
Request body (Content-Type: application/json)
{
  "typ": "..",          // member type (MTYPE domain)
  "did": "..",          // decentralized id
  "cty": "..",          // country code (ISO 3166-1 alpha-2)
  "desc": "..",         // organization name
  "email": "..",        // organization description
  "act": ..             // active flag (1|0)
}
```

- **Offboard Member** – Disables a Federation Member, reversing the onboarding operation but keeping the existing registration record.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM role.

```
PUT https://<public_address>/api/v1.0/members/{mid}/offboard
Path parameters:
• mid - member id
```

- **Delete Member** – Permanently removes an offboarded Federation Member from the system.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM role.

```
DELETE https://<public_address>/api/v1.0/members/{mid}
Path parameters:
• mid - member id
```

- **List Members** – Retrieves the list of Federation Members matching the given criteria.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM role.

```
GET https://<public_address>/api/v1.0/members
      ?act={..}&cty={..}&typ={..}&l={..}&o={..}
Query string parameters:
• act - active flag (optional)
        (0|1)          filter by active status
• cty - country code (optional)
        (string)       filter by country
• typ - member type (optional)
        (string)       filter by type
• l - page length (optional, to be used together with "o")
        (integer)      retrieve a maximum of n items
• o - page offset (optional, to be used together with "l")
        (integer)      retrieve items starting from n index (zero-based)
Response body (Content-Type: application/json)
[
  {                    // ARRAY ITEM
    "mid": "..",       // member id
    "typ": "..",       // member type (MTYPE domain)
    "did": "..",       // decentralized id
    "cty": "..",       // country code (ISO 3166-1 alpha-2)
    "name": "..",      // organization name
    "act": ..,         // active flag (1|0)
    "reg": ".."        // registration datetime (ISO 8601 UTC)
  },
  ..
]
```

- **Retrieve Member** – Retrieves all the information available for a given Federation Member.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM role.

```
GET https://<public_address>/api/v1.0/members/{mid}
Path parameters:
• mid - member id
Response body (Content-Type: application/json)
{
  "mid": "..",          // member id
  "typ": "..",          // member type (MTYPE domain)
  "did": "..",          // decentralized id
  "cty": "..",          // country code (ISO 3166-1 alpha-2)
  "name": "..",         // organization name
  "desc": "..",         // organization description
  "email": "..",        // contact point
  "act": ..,            // active flag (1|0)
  "reg": "..",          // registration datetime (ISO 8601 UTC)
  "aut": "..",          // registered by (uid)
  "chd": "..",          // last changed datetime (ISO 8601 UTC)
  "chr": ".."           // last changed by (uid)
}
```

- **Lookup Authority** – Checks if a given DID corresponds to a currently onboarded Federation Member.
  API Visibility: INTERNAL

```
GET http://<private_address>/gov/v1.0/authorities/{did}
Path parameters:
• did - decentralized id
Response body (Content-Type: application/json)
{
  "typ": "..",          // member type (MTYPE domain)
  "name": ".."          // organization name
}
```

4.2.2.2 Data Structures

`MTYPE` domain

- `RAUTH` - Root Authority (RA)
- `OAUTH` - Onboarding Authority (OA)
- `OTHER` - Any other role (e.g., federated dataspace without OA role)

`MEMBER` entity

```
mid STRING PK
did STRING NULL
typ STRING NOT NULL
cty STRING NOT NULL
name STRING NOT NULL
desc STRING NULL
email STRING NULL
act NOT NULL [1|0]
reg TS NOT NULL
aut STRING NOT NULL
chd TS NOT NULL
chr STRING NOT NULL
```

4.3 User Management

4.3.1 Description

The User Management component provides the Open API for the administration of a very basic pool of platform users. The definition of "platform user", in the FAME context, is the following: a *physical person* that is onboarded by an Onboarding Authority or Root Authority (see §4.2.1 for a short description of OA and RA federation member types), and thus receives a Verifiable Onboarding Credential (VOC) that can be stored in a FAME-compatible self-sovereign identity wallet and presented on request as a *proof of onboarding*.

Full-fledged OAs are *not* expected to use this component: they will typically have their own user management software and will deploy their own integrated solution for the issuance of VOCs. Instead, the User Management component is intended as a simple tool for the FAME *operational governance team* to register and onboard administrators and user support team members. However, it is worth noting that, for this to work, a Root Authority must be previously registered (in the Federation Management component) with a Decentralized Identifier (DID) that matches the one configured as the VOC issuer in the OID4VCI subsystem - for more details, see the installation guide in §5.2.2.

4.3.2 Technical Specification

4.3.2.1 Interfaces

- **Invite User** – Registers a new candidate user with the information provided by the caller and sends an invitation message to the specified contact point(s), thus starting the onboarding process.
  Note: the operation creates a new User record with system-assigned user identifier (UID) and invitation identifier (IID), CAN ("candidate") status and the attributes set by the caller; it then sends an invitation message to the contact point(s) specified for the user, containing a link to a web page (with the assigned IID as a query string parameter) that can be used to execute the Accept Invitation operation.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM role.

```
POST https://<public_address>/api/v1.0/users
Request body (Content-Type: application/json)
{
  "rol": "..",          // assigned role (see UROLE domain)
  "cty": "..",          // region (ISO 3166-1 alpha-2) (optional)
  "aff": "..",          // affiliation (PID) (optional)
  "fname": "..",        // first name
  "lname": "..",        // last name
  "email": "..",        // email contact
  "phone": ".."         // phone contact (optional)
}
Response body (Content-Type: application/json)
{
  "uid": "..",          // user id (assigned by system)
  "iid": ".."           // invitation id (assigned by system)
}
```

- **Accept Invitation** – Allows a candidate user to accept a previously-issued invitation and to receive a one-time-password (OTP) for starting the Verifiable Onboarding Credential (VOC) issuance sequence (see Claim Credential operation).
  Note: the operation selects the User record linked to the given IID and checks that is has CAN status; it then generates a random OTP and adds it to the record; finally, it sends a message, containing the OTP, to the contact point(s) saved with the record.

Note: the system maintains a counter that tracks how many times this operation is executed for the same invitation and blocks further executions after three runs; in case of need, this block can be removed by an administrator through the Renew Onboarding operation.
API Visibility: PUBLIC - Access Control: N/A (no authentication required).

```
PUT https://<public_address>/api/v1.0/invitations/{iid}/accept
Path parameters:
• iid - invitation id
```

- **Claim Credential** – Starts the Verifiable Onboarding Credential (VOC) issuance sequence for a previously-invited candidate user who is in possession of the correct OTP (see Accept Invitation operation), thus completing the onboarding process.
  Note: the operation selects the User record with the given IID, verifies that it has INV status and that the given OTP matches the value saved with the record; it then changes the record status to ONB ("onboarded"), clears the saved OTP, and activates the VOC Issuance process for the target user on the OID4VCI subsystem (see last bullet point); finally, it sends a notification message to the contact point(s) specified for the user, and returns the *pre-authorized code* that must be used to retrieve the VOC[19].
  Note: as the saved OTP value is cleared, this operation is not repeatable; in case of need, this block can be removed by an administrator through the Renew Onboarding operation.
  API Visibility: PUBLIC - Access Control: N/A (no authentication required).

```
PUT https://<public_address>/api/v1.0/invitations/{iid}/claim
Path parameters:
• iid - invitation id
Request body (Content-Type: application/json)
{
  "otp": ".."           // one-time-password
}
Response body (Content-Type: application/json)
{
  "code": ".."          // OID4VIC pre-authorized code
}
```

- **Update User** – Updates a User record with the given data.
  Note: attributes that are included in the user's VOC cannot be changed. This implies that a change of role, affiliation and/or country requires the user to go through a new onboarding process, at the end of which they will be assigned a new UID and VOC.
  Note: all attributes provided by the caller are optional, missing ones will not be changed.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM or SUP role.

```
PUT https://<public_address>/api/v1.0/users/{uid}
Path parameters:
• uid - user id
Request body (Content-Type: application/json)
{
  "fname": "..",        // first name
  "lname": "..",        // last name
  "email": "..",        // email contact point
  "phone": ".."         // phone contact
}
```

[19] This value is a string that represents a *credential offer* according to the *pre-authorized code flow* defined by the *OpenID for Verifiable Credential Issuance* (OID4VCI) standard (see https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html). Typically, the string will be presented as a QR-Code on the user-facing web page: the user will then activate their FAME Identity Wallet and scan the QR-Code to receive the VOC issued by the FAME authority (OA or RA).

- **Renew Onboarding** – Allows an administrator to reset a (possibly stuck) onboarding process or to start a new onboarding process for an existing (possibly offboarded) user, by issuing a new invitation.
  Note: the operation selects the User record identified by the UID, regardless of status; it then sets the status to CAN, overwrites the saved IID with a new system-generated value and resets the execution counter of the Accept Invitation operation to zero; finally, it sends an invitation message to the contact point(s) specified for the user, containing a link to a web page that can be used to execute the Accept Invitation operation, with the new IID as a parameter.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM role.

```
PUT https://<public_address>/api/v1.0/users/{uid}/onboard
Path parameters:
• uid - user id
Response body (Content-Type: application/json)
{
  "iid": ".."           // new invitation id (re-assigned by system)
}
```

- **Offboard User** – Revokes the onboarding of a user.
  Note: the operation selects the User record identified by the UID and checks that is has ONB status; it then sets the new status to OFB ("offboarded") and activates the VOC Revocation process for the target user on the dedicated GOV subsystem (see §4.1); finally, it sends a notification message to the contact point(s) specified for the user.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM role.

```
PUT https://<public_address>/api/v1.0/users/{uid}/offboard
```

- **Delete User** – Permanently removes a User record from the system.
  Note: the operation selects the User record identified by the UID and checks that is has *not* ONB status[20]; it then deletes the record from the registry.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM role.

```
DELETE https://<public_address>/api/v1.0/users/{uid}
```

- **List Users** – Retrieves the list of User records matching the given criteria.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM or SUP role.

```
GET https://<public_address>/api/v1.0/users
      ?sta={..}&rol={..}&cty={..}&aff={..}&l={..}&o={..}
Query string parameters:
• sta - status (optional)
        (USTAT domain) filter by status
        multiple values can be provided using the "|" separator
• rol - role (optional)
        (UROLE domain) filter by role
        multiple values can be provided using the "|" separator
• cty - country code (optional)
        (string)       filter by region
• aff - provenance id (optional)
        (string)       filter by affiliation
• l - page length (optional, to be used together with "o")
        (integer)      retrieve a maximum of n items
```

[20] Onboarded users must first be offboarded.

```
• o - page offset (optional, to be used together with "l")
        (integer)      retrieve items starting from n index (zero-based)
Response body (Content-Type: application/json)
[
  {                     // ARRAY ITEM
    "uid": "..",        // user id
    "iid": "..",        // invitation id
    "sta": "..",        // status (USTAT domain)
    "rol": "..",        // role (UROLE domain)
    "cty": "..",        // country code (ISO 3166-1 alpha-2)
    "aff": "..",        // affiliation (PID)
    "fname": "..",      // first name
    "lname": ".."       // last name
  },
  ..
]
```

- **Retrieve User** – Retrieves all the information available for a given User.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM or SUP role.

```
GET https://<public_address>/api/v1.0/users/{uid}
Path parameters:
• uid - user id
Response body (Content-Type: application/json)
{
  "uid": "..",          // user id
  "iid": "..",          // invitation id
  "sta": "..",          // status (USTAT domain)
  "rol": "..",          // assigned role (see UROLE domain)
  "cty": "..",          // region (ISO 3166-1 alpha-2)
  "aff": "..",          // affiliation (PID)
  "fname": "..",        // first name
  "lname": "..",        // last name
  "email": "..",        // email contact
  "phone": ".."         // phone contact
}
```

- ***OID4VCI subsystem*** – This is not a FAME-specific service endpoint, but rather a family of generic services that implement the Issuer side of the *OpenID for Verifiable Credential Issuance* (OID4VCI) standard. If using the FAME Identity Wallet mobile app, which will be provided as part of the FAME AAI package (see deliverable D3.1 Secure Federated Data Management v2), there is no need for a user to integrate any additional software with this API: the mobile app has the capability, out-of-the-box, of exploiting the Platform's OID4VCI subsystem in order to receive Verifiable Onboarding Credentials. For this reason, in this document we don't provide the technical specs of the services: for this level of detail, you can refer to the official documentation of the OID4VCI standard at https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html. Note that the protocol implemented here is the "pre-authorized code flow", where the initial user interaction that leads to the issuance of the pre-authorization code is done through the Invite User, Accept Invitation and Claim Credential operations documented above.

4.3.2.2 Data Structures

`UROLE` domain

- `ADM` – Platform administrator
- `SUP` – User support (helpdesk) operator
- `ENT` – Large enterprise employee
- `SME` – Small/medium enterprise employee
- `GOV` – Government organization member (civil servant)
- `LEA` – Law enforcement agency member
- `NOP` – No-profit organization member
- `RES` – Research organization member
- `EDU` – Educational organization member
- `STU` – Student
- `USR` – Generic user (none of the above)

`USTAT` domain

- `CAN` – Candidate
- `ONB` – Onboarded
- `OFB` – Offboarded

`USER` entity

```
uid STRING PK           // user id
iid STRING              // invitation id
sta STRING NOT NULL     // status (USTAT domain)
acc INT                 // count of "accept invitation" executions
otp STRING              // one-time-password to unlock "claim credential"
rol STRING NOT NULL     // role (UROLE domain)
cty STRING              // country code (ISO 3166-1 alpha-2)
aff STRING              // affiliation (PID)
fname STRING            // first name
lname STRING NOT NULL   // last name
email STRING NOT NULL   // email contact (will receive notifications)
phone STRING            // SMS contact (will receive notifications)
reg STRING NOT NULL     // registration datetime (ISO 8601 UTC)
aut STRING NOT NULL     // authorized by (UID)
chd STRING NOT NULL     // last change datetime (ISO 8601 UTC)
chr STRING NOT NULL     // last change by (UID)
```

4.4 User Support

4.4.1 Description

The User Support component works on two levels.

Firstly, it implements a ticketing system that serves as the central hub for all user support activities: platform users can ask for support on FAME-related issues by opening a Support Ticket, and the support team will be able to follow up and track the status of the issue through the same system. This mediation role is extremely important due to the privacy-friendly way in which the FAME Platform is designed, as users are not supposed to disclose their contact information - or even their email address - with anyone outside of their Onboarding Authority.

Secondly, the component implements a User Request queue. Open API operations that are processed asynchronously (e.g., Enroll / Disenroll Account operations in the Trading Support component - see §4.5.2.1) will use the queue as a messaging channel: the user submits an execution request, the request

is dispatched to the Platform service in charge of execution, the service updates the request by adding the execution's outcome, and the user checks the outcome.

4.4.2 Technical Specification

4.4.2.1 Interfaces

- **Open Ticket** – Creates a new Support Ticket in the system, with a system-assigned ID, status set to “pending” and the caller's UID as "owner".
  API Visibility: PUBLIC - Access Control: any authenticated user

```
POST https://<public_address>/api/v1.0/tickets
Request body (Content-Type: application/json)
{
  "iss": "..",          // issue type (TTYPE domain)
  "des": ".."           // issue description
}
Response body (Content-Type: application/json)
{
  "sid": ".."           // support ticket id (assigned by system)
}
```

- **Receive Ticket** – Sets the status of a "pending", "processing" or "suspended" Support Ticket to “processing” and sets the caller's UID as "assignee".
  API Visibility: PUBLIC - Access Control: authenticated user with SUP or ADM role.

```
PUT https://<public_address>/api/v1.0/tickets/{sid}
Path parameters:
• sid - support ticket id
Request body (Content-Type: application/json)
{
  "opt": "RECEIVE"      // operation type: receive
}
```

- **Reply to Ticket** – Appends an entry to the history of a Support Ticket with "processing" status.
  API Visibility: PUBLIC - Access Control: any authenticated user; additional restrictions:
  - if the user has SUP role, they shall be the ticket's current assignee;
  - if the user has neither SUP nor ADM role, they shall be the ticket owner; in this case, the "int" field of the request body, if set, is ignored by the system.

```
PUT https://<public_address>/api/v1.0/tickets/{sid}
Path parameters:
• sid - support ticket id
Request body (Content-Type: application/json)
{
  "opt": "REPLY",       // operation type: reply
  "pum": "..",          // public message appended to the log
  "inn": ".."           // internal note appended to the log (optional)
}
```

- **Suspend Ticket** – Sets the status of a "processing" Support Ticket to “suspended”; additionally, if a public and/or internal message are provided by the caller, they are appended to the history of the ticket.
  API Visibility: PUBLIC - Access Control: authenticated user with SUP or ADM role; additional restrictions: if the user has SUP role, they shall be the ticket's current assignee.

```
PUT https://<public_address>/api/v1.0/tickets/{sid}
Path parameters:
• sid - support ticket id
```

```
Request body (Content-Type: application/json)
{
  "opt": "SUSPEND",     // operation type: suspend
  "pum": "..",          // public message appended to the log (optional)
  "inm": ".."           // internal note appended to the log (optional)
}
```

- **Close Ticket** – Sets the status of a "pending" or "processing" Support Ticket to “closed”, and its "outcome" to the given value; additionally, if a public and/or internal message are provided by the caller, they are appended to the history of the ticket.
  API Visibility: PUBLIC - Access Control: authenticated user with SUP or ADM role.

```
PUT https://<public_address>/api/v1.0/tickets/{sid}
Path parameters:
• sid - support ticket id
Request body (Content-Type: application/json)
{
  "opt": "CLOSE",       // operation type: close
  "out": "..",          // outcome (ROUT domain)
  "pum": "..",          // public message appended to the log (optional)
  "inm": ".."           // internal note appended to the log (optional)
}
```

- **Reopen Ticket** – Sets the status of a "closed" Support Ticket to “processing” and sets the caller's UID as "assignee"; additionally, if a public and/or internal message are provided by the caller, they are appended to the history of the ticket.
  API Visibility: PUBLIC - Access Control: authenticated user with SUP or ADM role.

```
PUT https://<public_address>/api/v1.0/tickets/{sid}
Path parameters:
• sid - support ticket id
Request body (Content-Type: application/json)
{
  "opt": "REOPEN",      // operation type: reopen
  "pum": "..",          // public message appended to the log (optional)
  "inm": ".."           // internal note appended to the log (optional)
}
```

- **Update Ticket** – Updates a Support Ticket with the given data, regardless of the workflow state.
  Note: all attributes provided by the caller are optional, missing ones will not be changed.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM role.

```
PUT https://<public_address>/api/v1.0/tickets/{sid}
Path parameters:
• sid - support ticket id
Request body (Content-Type: application/json)
{
  "opt": "UPDATE",      // operation type: update
  "sta": "..",          // new status (TSTAT domain)
  "ass": "..",          // new assignee (uid)
  "out": ".."           // new outcome (ROUT domain)
}
```

- **List Tickets** – Retrieves the list of Support Tickets matching the given criteria.

API Visibility: PUBLIC - Access Control: any authenticated user; note: if user has neither SUP nor ADM role, then the returned list will be filtered so that only tickets owned by the caller are included

```
GET https://<public_address>/api/v1.0/tickets
      ?iss={..}&sta={..}&out={..}&own={..}&ass={..}&l={..}&o={..}
Query string parameters:
```

- iss - issue type (optional)
  (TTYPE domain) filter by type of issue
  multiple values can be provided using the "|" separator
- sta - ticket status (optional)
  (TSTAT domain) filter by processing status
  multiple values can be provided using the "|" separator
- out - request outcome (optional)
  (ROUT domain) filter by solution
  multiple values can be provided using the "|" separator
- own - ticket owner (optional)
  (string) filter by uid of ticket opener
- ass - ticket assignee (optional)
  (string) filter by uid of operator in charge
- l - page length (optional, to be used together with "o")
  (integer) retrieve a maximum of *n* items
- o - page offset (optional, to be used together with "l")
  (integer) retrieve items starting from *n* index (zero-based)

```
Response body (Content-Type: application/json)
[
  {                     // ARRAY ITEM
    "sid": "..",        // support ticket id
    "sub": "..",        // submission (ISO 8601 UTC)
    "upd": ".."         // last update (ISO 8601 UTC)
    "own": "..",        // ticket owner (uid)
    "iss": "..",        // issue type (TTYPE domain)
    "sta": "..",        // effective ticket status (TSTAT domain)
    "ass": "..",        // effective assignee (uid)
    "out": ".."         // effective request outcome (ROUT domain)
  },
  ..
]
```

- **Retrieve Ticket** – Retrieves all the information available for a given Support Ticket.
  API Visibility: PUBLIC - Access Control: any authenticated user; additional restrictions: if the user has neither SUP nor ADM role, they shall be the ticket owner.

```
PUT https://<public_address>/api/v1.0/tickets/{sid}
Path parameters:
```

- sid - support ticket id

```
Response body (Content-Type: application/json)
{
  "sid": "..",          // support ticket id
  "sub": "..",          // submission (ISO 8601 UTC)
  "upd": ".."           // last update (ISO 8601 UTC)
  "own": "..",          // ticket owner (uid)
  "iss": "..",          // issue type (TTYPE domain)
  "des": "..",          // issue description
  "sta": "..",          // effective ticket status (TSTAT domain)
  "ass": "..",          // effective assignee (uid)
  "out": "..",          // effective request outcome (ROUT domain)
  "log": [              // ticket event history (descending order on "evn")
```

```
    {                   // ARRAY ITEM
      "evn": ..,        // event prog number (1-based)
      "ets": "..",      // event time (ISO 8601 UTC)
      "opt": "..",      // operation type (OTYPE domain)
      "esr": "..",      // acting operator (uid)
      "sta": "..",      // current ticket status (TSTAT domain)
      "ass": "..",      // current assignee (uid)
      "out": "..",      // current request outcome (ROUT domain)
      "pum": "..",      // public message associated with event
      "inm": ".."       // internal note associated with event
    },
    ..
  ]
}
```

- **Check Request status** – Verifies the status of a previously submitted User Request.
  API Visibility: PUBLIC - Access Control: any authenticated user; additional restrictions: if the caller has no ADM role, then they shall be the owner of the request ("requestor").

```
GET https://<public_address>/api/v1.0/requests/{rid}
Response body (Content-Type: application/json)
{
  "finalized": "..",    // timestamp of finalization (ISO 8601 UTC)
  "status": "..",       // status (PENDING|PROCESSED|EXPIRED|ERROR)
  "message": ".."       // human-readable text
}
```

- **Enqueue Request** – Creates a new "pending" User Request in the system.
  API Visibility: INTERNAL

```
POST http://<private_address>/gov/v1.0/rqueue
Request body (application/json)
{
  "rid": "..",          // unique id assigned by the caller
  "requestor": "..",    // uid of request owner
  "payload": {..}       // application-specific payload (any JSON literal)
}
```

- **Retrieve Request** – Retrieves the given User Request record.
  API Visibility: INTERNAL

```
GET http://<private_address>/gov/v1.0/rqueue/{rid}
Path parameters:
```

  - `rid - request id`

```
Response body (Content-Type: application/json)
{
  "requestor": "..",    // uid of user who submitted the request
  "enqueued": "..",     // timestamp of submission (ISO 8601 UTC)
  "payload": {..},      // application-specific payload
  "finalized": "..",    // timestamp of finalization (ISO 8601 UTC)
  "status": "..",       // status (PENDING|PROCESSED|EXPIRED|ERROR)
  "message": ".."       // human-readable text
}
```

- **Finalize Request** – Updates a User Request record with the given data and status.
  API Visibility: INTERNAL

```
PUT http://<private_address>/gov/v1.0/rqueue/{rid}
Path parameters:
```

- rid - request id

```
Request body (application/json)
{
  "status": "..",       // updated status (PROCESSED|ERROR)
  "message": ".."       // human-readable text (optional)
}
```

4.4.2.2 Data Structures

`TTYPE` domain

- `EXCHANGE` - Related to payment token exchange
- `TRADING` - Related to asset trading
- `TECH` - Related to other technical issues
- `OTHER` - Related to other non-technical issues

`OTYPE` domain

- `RECEIVE`
- `REPLY`
- `SUSPEND`
- `CLOSE`
- `REOPEN`
- `UPDATE`

`TSTAT` domain

- `PENDING`
- `PROCESSING`
- `SUSPENDED`
- `CLOSED`

`ROUT` domain

- `SOLVED` - Issue was solved, at least partially
- `UNSOLVABLE` - Issue could not be solved
- `REJECTED` - Issue was out of scope

`TICKET` entity

```
sid STRING PK           // support ticket id
sub STRING NOT NULL     // submission (ISO 8601 UTC)
upd STRING NOT NULL     // last update (ISO 8601 UTC)
own STRING NOT NULL     // ticket owner (uid)
iss STRING NOT NULL     // issue type (TTYPE domain)
des STRING NOT NULL     // issue description
sta STRING NOT NULL     // effective ticket status (TSTAT domain)
ass STRING              // effective assignee (uid)
out STRING              // effective request outcome (ROUT domain)
```

`EVENTS` entity

```
sid STRING PK           // support ticket id
evn INT PK              // event id
ets STRING FK           // event time (ISO 8601 UTC)
opt STRING NOT NULL     // operation type (OTYPE domain)
esr STRING NOT NULL     // acting operator (uid)
sta STRING NOT NULL     // current ticket status (TSTAT domain)
ass STRING              // current assignee (uid)
out STRING              // current request outcome (ROUT domain)
pum STRING              // public message associated with event
```

```
inm STRING              // internal note associated with event
```

4.5 Trading Support

4.5.1 Description

To explain the role of the Trading Support component, some background is needed.

Trading transactions in FAME are entirely managed on the FAME Blockchain Infrastructure. *Trading parties* - i.e., buyers and sellers of digital assets - must use a *trading account* - i.e., a plain Ethereum-style blockchain account that has been granted permission to operate on the FAME Blockchain Infrastructure - for executing asset-related transactions, sending / receiving transaction-related payments, and receiving Non-Fungible Tokens (NFT) that represent *rights* on an asset derived from a transaction (e.g., temporary access granted to some digital content).

Transaction-related payments are enabled by *payment tokens*. The FAME payment token is named **FDE coin** (FAME Digital Euro), as it has a nominal 1-to-1 equivalence with the EUR currency[21]. Trading parties can *acquire* FDE coins by buying them from the FAME support desk. This typically requires the user to transfer the equivalent amount of "real money" to a bank account managed by the FAME support desk. The same process also works in reverse: users can *redeem* FDE coins they already own (i.e., exchange them with the equivalent amount of "real money") and have their bank account credited by the FAME support desk. Overall, these processes are under the control of the *token exchange service*.

Within this context, the Trading Support component has the following capabilities:

- Granting or revoking the permission to operate on individual trading accounts.
- Tracking the ownership of trading accounts, linking each of them to the correct trading party.
- Provide insight into the current and historical state of trading accounts (e.g., ownership, balance, past transactions).
- Manage the token exchange service, allowing the FAME support desk to move FDE coins in and out of any trading account.

Regarding token exchange service, there is one important caveat: FDE coins are only managed as a virtual currency for testing purposes in the scope of the FAME project. *No real money is involved, either explicitly or implicitly*.

4.5.2 Technical Specification

The FDE coin (see §4.5.1) is implemented as a standard ERC-20 contract[22], named **PaymentToken**, which is deployed on the FAME Blockchain Infrastructure. The diagram in Figure 9 shows the structure of the PaymentToken contract and the interfaces it provides.

[21] This makes FDE a *stablecoin*, in cryptocurrency jargon. But see also the closing note in this chapter.
[22] See https://ethereum.org/en/developers/docs/standards/tokens/erc-20/

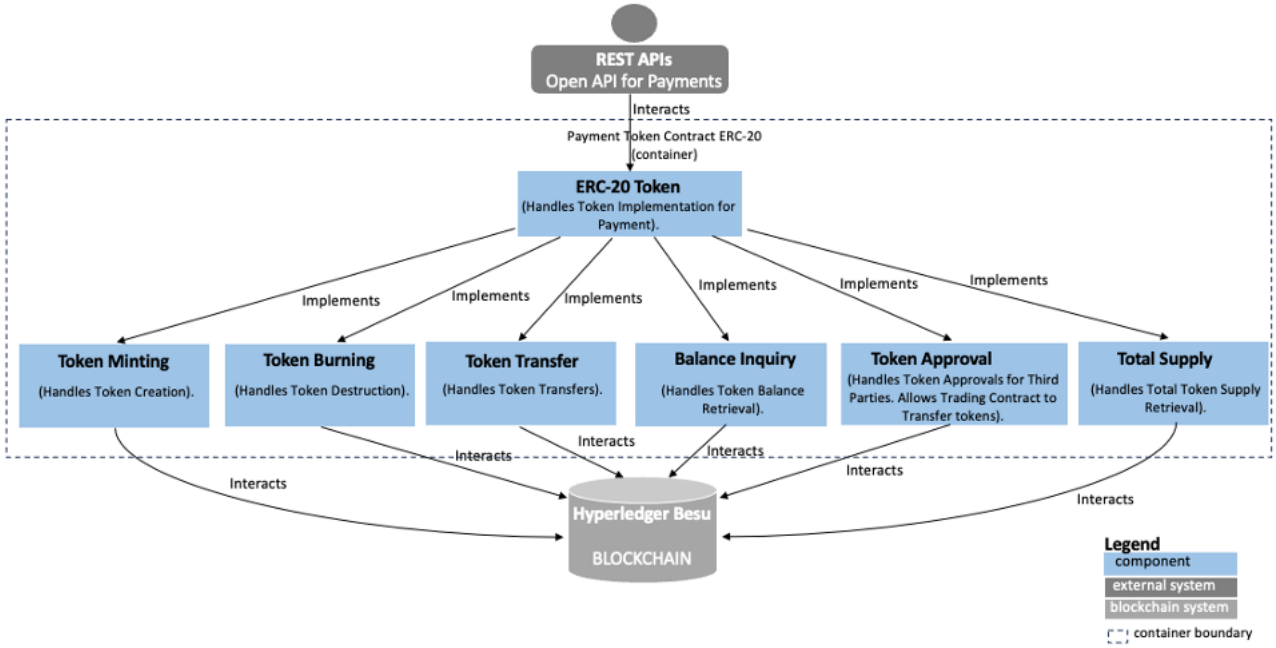

Figure 9 - ERC-20 payment token contract

The PaymentToken contract includes the following components:

1. **ERC-20 Token**: This is the main component that implements the standard ERC-20 token for payment.
2. **Mint Payment Token**: This component handles the minting X amount of payment tokens to the specified user
3. **Burn Payment Token**: This component handles the burning X amount of payment tokens of the specified user
4. **Token Transfer**: This component handles the transfer of tokens between addresses.
5. **Balance Inquiry**: This component provides information about token balances.
6. **Token Approval**: This component handles token approval for third parties. It allows the Trading Contract to transfer assets.
7. **Total Supply**: This component provides information about the total token supply.

Whereas the PaymentToken smart contract implements the ERC20 standard token functions, the **FameGovernance** smart contract supports registration of allowed payment token and its symbol and token minting, burning and balance query operations and **FameBourse** smart contracts supports data buying/selling operations with the allowed payment tokens. Moreover, a REST API with internal visibility is also provided for easy integration of the payment tokens subsystem with the rest of the platform.

4.5.2.1 Interfaces

Note on currency token amounts - The amount of currency tokens, when used as an input parameter or output value of an Open API operation, is always expressed in *FDE coin units*, which is a number with a *fixed 2-digit decimal part*. This is similar to how EUR or USD currency amounts are commonly displayed. However, token accounting on the blockchain is based on integer numbers, with the coin divided into atomic units representing an *arbitrary fraction* (e.g., one billionth) of the whole. This means that when an amount is returned by an Open API call, rounding to the nearest 2-digit decimal

cents value may be applied. The Open API operations that deal with coin units are Credit Account, Debit Account, Retrieve Balance, List Transactions.
On the other hand, all internal API operations that deal with generic currency tokens always work at the lowest level, using atomic units (i.e., the smallest part of the coin that is not divisible). This is the case for Retrieve Balance for Token and Mint / Burn Tokens.

- **Enroll Account** – Registers a Blockchain account as a Trading Account owned by the caller.
  API Visibility: PUBLIC - Access Control: any authenticated user
  Asynchronous call: the caller can verify the outcome by retrieving the User Request identified in the response (see Retrieve Request operation).

```
POST https://<public_address>/api/v1.0/submissions/accounts
Request body (Content-Type: application/json)
{
  "opid": "ENROLL",     // operation id: enroll
  "bid": ".."           // blockchain account id
}
Response body (Content-Type: application/json)
{
  "rid": ".."           // user request id (assigned by system)
}
```

- **Disenroll Account** – Disables a given Trading Account, reversing the enrollment operation.
  API Visibility: PUBLIC - Access Control: any authenticated user; additional restrictions: if user has no ADM role, then they shall be the owner of the account.
  Asynchronous call: the caller can verify the outcome by retrieving the User Request identified in the response (see *Retrieve Request* operation).

```
POST https://<public_address>/api/v1.0/submissions/accounts
Request body (Content-Type: application/json)
{
  "opid": "DISENROLL",  // operation id: disenroll
  "bid": ".."           // blockchain account id
}
Response body (Content-Type: application/json)
{
  "rid": ".."           // user request id (assigned by system)
}
```

- **Credit Account** – Mints a given amount of FDE coins and transfers them to a given Trading Account.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM or SUP role.

```
PUT https://<public_address>/api/v1.0/accounts/{tid}/credit
Path parameters:
• tid - trading account id
Request body (Content-Type: application/json)
{
  "amount": ..          // number of coins to mint (two-decimal number)
}
```

- **Debit Account** – Burns a given amount of FDE coins from a given Trading Account.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM or SUP role.

```
PUT https://<public_address>/api/v1.0/accounts/{tid}/debit
Path parameters:
• tid - trading account id
Request body (Content-Type: application/json)
```

```
{
  "amount": ..          // number of coins to burn (two-decimal number)
}
```

- **List Accounts** – Retrieves the list of Trading Accounts matching the given criteria.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM or SUP role.

```
GET https://<public_address>/api/v1.0/accounts
      ?pid={..}&uid={..}&cty={..}&act={..}&frm={..}&til={..}&l={..}&o={..}
Query string parameters:
• pid - provenance id (optional)
        (string)       filter by trader affiliation
• uid - user id (optional)
        (string)       filter by trader
• cty - country code (optional)
        (string)       filter by trader's country
• act - active flag (optional)
        (0|1)          filter by active status
• frm - from date (optional)
        YYYYMMDD       filter by registration date: on this date or later
• til - until date (optional)
        YYYYMMDD       filter by registration date: on this date or before
• l - page length (optional, to be used together with "o")
        (integer)      retrieve a maximum of n items
• o - page offset (optional, to be used together with "l")
        (integer)      retrieve items starting from n index (zero-based)
Response body (Content-Type: application/json)
[
  {                    // ARRAY ITEM
    "tid": "..",       // trading account id
    "aff_pid": "..",   // trader affiliation entity id (can be empty)
    "trd_uid": "..",   // trader id
    "trd_cty": "..",   // trader's country
    "act": ..,         // trading account active flag (0|1)
    "reg": ".."        // trading account registration (ISO 8601 UTC)
  },
  ..
]
```

- **Retrieve Account** – Retrieves all the information available for a given Trading Account.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM or SUP role.

```
GET https://<public_address>/api/v1.0/accounts/{tid}
Path parameters:
• tid - trading account id
Response body (Content-Type: application/json)
{
  "tid": "..",         // trading account id
  "aff_pid": "..",     // trader affiliation entity id (can be empty)
  "aff_name": "..",    // trader affiliation entity name (can be empty)
  "trd_uid": "..",     // trader id
  "trd_cty": "..",     // trader's country
  "act": ..,           // trading account active flag (0|1)
  "reg": "..",         // trading account registration (ISO 8601 UTC)
  "chd": "..",         // record last changed (ISO 8601 UTC)
  "chr": ".."          // record last changer (uid)
}
```

- **List Transactions** – Retrieves the list of transactions related to the transfer of FDE coins affecting a given Trading Account, matching the given criteria.
  API Visibility: PUBLIC - Access Control: any authenticated user; additional restrictions: if the caller has no ADM or SUP role, then they shall be the owner of the account.

```
GET https://<public_address>/api/v1.0/accounts/{tid}/transactions
      ?typ={..|..}&frm={..}&til={..}&l={..}&o={..}
Path parameters:
• tid - trading account id
Query string parameters:
• typ - transaction type (optional)
        "0"            coins acquired (Credit Account op)
        "1"            coins redeemed (Debit Account op)
        "2"            coins spent in a trading transaction
        "3"            coins received from a trading transaction
        multiple values can be provided using the "|" separator
• frm - from date (optional)
        YYYYMMDD       filter by execution date: on this date or later
• til - until date (optional)
        YYYYMMDD       filter by execution date: on this date or before
• l - page length (optional, to be used together with "o")
        (integer)      retrieve a maximum of n items
• o - page offset (optional, to be used together with "l")
        (integer)      retrieve items starting from n index (zero-based)
Response body (Content-Type: application/json)
[
  {                    // ARRAY ITEM
    "id": "..",        // blockchain transaction id
    "ts": "..",        // execution (ISO 8601 UTC)
    "typ": ..,         // transaction type (0|1|2|3)
    "oid": "..",       // offering id (only if trading transaction)
    "amount": ..       // amount transferred (two-decimal number)
  },
  ..
]
```

- **Retrieve Balance** – Retrieves the amount of FDE coins currently owned by a given Trading Account.
  API Visibility: PUBLIC - Access Control: any authenticated user; additional restrictions: if the caller has no ADM or SUP role, then they shall be the owner of the account.

```
GET https://<public_address>/api/v1.0/accounts/{tid}/balance
Path parameters:
• tid - trading account id
Response body (Content-Type: application/json)
{
  "balance": ..        // amount available (two-decimal number)
}
```

- **List Traders** – Retrieves the list of users that own one or more Trading Accounts and match the given criteria.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM or SUP role.

```
GET https://<public_address>/api/v1.0/traders
      ?pid={..}&cty={..}&l={..}&o={..}
Query string parameters:
• pid - provenance id (optional)
        (string)       filter by trader affiliation
```

```
• cty - country code (optional)
        (string)       filter by trader's country (country code)
• l - page length (optional, to be used together with "o")
        (integer)      retrieve a maximum of n items
• o - page offset (optional, to be used together with "l")
        (integer)      retrieve items starting from n index (zero-based)
Response body (Content-Type: application/json)
[
  {                    // ARRAY ITEM
    "uid": "..",       // trader id
    "cty": "..",       // trader's country
    "aff_pid": "..",   // trader affiliation entity id (can be empty)
    "aff_name": "..",  // trader affiliation entity name (can be empty)
    "reg": "..",       // trader's registration (ISO 8601 UTC)
    "acount": ..       // number of active trading accounts
  },
  ..
]
```

- **Retrieve Trader** – Retrieves all the information available for a given user that owns one or more Trading Accounts.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM or SUP role.

```
GET https://<public_address>/api/v1.0/traders/{uid}
Path parameters:
• uid - user id
Response body (Content-Type: application/json)
{
  "aff_pid": "..",      // affiliation entity id (can be empty)
  "aff_name": "..",     // affiliation entity name (can be empty)
  "cty": "..",          // country
  "reg": "..",          // registration (ISO 8601 UTC)
  "accounts": [         // owned trading accounts (array)
    {                   // ARRAY ITEM
      "tid": "..",      // trading account id
      "act": ..,        // trading account active flag (0|1)
      "reg": ".."       // trading account registration (ISO 8601 UTC)
    },
    ..
  ]
}
```

- **List Affiliations** – Retrieves the list of Trusted Sources linked to Trading Accounts.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM or SUP role.

```
GET https://<public_address>/api/v1.0/affiliations
      ?l={..}&o={..}
Query string parameters:
• l - page length (optional, to be used together with "o")
        (integer)      retrieve a maximum of n items
• o - page offset (optional, to be used together with "l")
        (integer)      retrieve items starting from n index (zero-based)
Response body (Content-Type: application/json)
[
  {                    // ARRAY ITEM
    "pid": "..",       // entity id
    "name": "..",      // entity name
    "tcount": ..,      // number of affiliated traders
    "acount": ..       // number of active trading accounts
```

```
  },
  ..
]
```

- **Retrieve Affiliation** – Retrieves all the information available for a given Trusted Source linked to Trading Accounts.
  API Visibility: PUBLIC - Access Control: authenticated user with ADM or SUP role.

```
GET https://<public_address>/api/v1.0/affiliations/{pid}
Path parameters:
• pid - provenance id
Response body (Content-Type: application/json)
{
  "name": "..",          // entity name
  "traders": [           // affiliated traders (array)
    {                    // ARRAY ITEM
      "uid": "..",       // trader id
      "cty": "..",       // trader country
      "reg": "..",       // trader registration (ISO 8601 UTC)
      "acount": ..       // number of active trading accounts
    },
    ..
  ]
}
```

- **Lookup Ownership** – Given the identifier of a currently enrolled Trading Account, returns the linked user and affiliation.
  API Visibility: INTERNAL

```
GET http://<private_address>/gov/v1.0/owners/{tid}
Path parameters:
• tid - trading account id
Response body (Content-Type: application/json)
{
  "uid": "..",           // trader id
  "pid": ".."            // affiliation id (can be empty)
}
```

The following two API operations are only used internally: they are a convenient wrapper around the FameGovernance smart contract functions, which are listed separately in §4.5.2.2. Once a currency token has been registered with its symbol, query of balances of accounts as well as minting and burning of tokens can be done using its symbol.

- **Retrieve Balance for Token** – Retrieves the number of tokens of a given type that are currently owned by given Trading Account.
  API Visibility: INTERNAL

```
GET http://<private_address>/gov/v1.0/trading-accounts/{tid}?t={..}
Path parameters:
• tid - trading account id
Query string parameters:
• t - token symbol
```

- **Mint / Burn Tokens** – Mints a given amount of a given token and transfers them to a given Trading Account OR burns a given amount of a given token from a given Trading Account.
  API Visibility: INTERNAL

```
PUT http://<private_address>/gov/v1.0/trading-accounts/{tid}
```

```
Path parameters:
• tid - trading account id
Request body (Content-Type: application/json)
{
  "operation": "..",    // type of operation (MINT|BURN)
  "token": "..",        // token symbol
  "amount": ..          // number of tokens to mint or burn (int)
}
```

The following family of API operations is only used internally: they are a convenient wrapper around the FameBourse smart contract functions, which can extract the trading transaction history from the blockchain. The structure of the calls is such that first the number of trade pairs can be obtained using call `getNumberOfTradePairs`. Then using `getIthTradePair` function, trade pairs can be enumerated. For each trade pair, buy/sell transactions can be enumerated either for each individual trader using `gethIthTraderSell`/`getIthTraderBuy` or for all traders using `gethIthBuy`/`getIthSell` functions. For each trade pair, total number and volume of buy and sell transactions can also be retrieved using `getTotalNumVolBuys`/`getTotalNumVolSells` functions. Due to the peculiar use of these operations, their path does *not* follow the standard REST-over-HTTP guidelines.

API Visibility: INTERNAL

- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getNumberOfTradePairs`
- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getIthTradePair/{ithp}`
- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getTotalNumVolBuys/{ta1}/{ti1}/{ta2}/{ti2}`
- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getTotalNumVolSells/{ta1}/{ti1}/{ta2}/{ti2}`
- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getNumberOfTraderBuys/{tid}/{ta1}/{ti1}/{ta2}/{ti2}`
- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getNumberOfTraderSells/{tid}/{ta1}/{ti1}/{ta2}/{ti2}`
- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getIthTraderSell/{ith}/{tid}/{ta1}/{ti1}/{ta2}/{ti2}`
- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getIthTraderBuy/{ith}/{tid}/{ta1}/{ti1}/{ta2}/{ti2}`
- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getNumberOfSells/{ta1}/{ti1}/{ta2}/{ti2}`
- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getNumberOfBuys/{ta1}/{ti1}/{ta2}/{ti2}`
- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getIthBuy/{ith}/{ta1}/{ti1}/{ta2}/{ti2}`
- `GET http://<private_address>/gov/v1.0/retrieve-trading-history /getIthSell/{ith}/{ta1}/{ti1}/{ta2}/{ti2}`

```
Path parameters:
   o  tid - trading account id
   o  ithp - ith trade pair
   o  ta1 - first token address in trade pair
   o  ti2 - first token index in trade pair
   o  ta2 - second token address in trade pair
   o  ti2 - second token index in trade pair
   o  ith - ith buy or sell transaction
```

4.5.2.2 Smart Contract Functions

The smart contract functions of the FameGovernance are listed below. Even though only a token with the FDE symbol is currently being used in the FAME project, FameGovernance has been developed with multi-currency and utility token uses in mind, by allowing general ERC-20 and ERC-1155 token contracts to be used as currency coins.

- `setFameBourse(address bourseaddr)`
  `public onlyOwner`
- `getCoinBalance(address traderid, string memory tokensymbol)`
  `public view returns(uint)`
- `registerCoin(address cointokenaddr, uint tokindx, string coinsymbol)`
  `public onlyOwner returns(uint)`
- `mintCoin(address to_traderid, string coinsymbol, uint amount)`
  `public returns(bool)`
- `burnCoin(address from_traderid, string coinsymbol, uint amount)`
  `public returns(bool)`
- `getNoofCoins()`
  `public view returns(uint)`
- `getCoinSymbol(uint coinno)`
  `public view returns(string)`
- `getCoinAddress(string coinsymbol)`
  `public view returns (address, uint)`
- `getCoinNo(address coinaddr, uint tokindx)`
  `public view returns(uint)`

4.5.2.3 Data Structures

Here we only document the Trading Accounts database schema. The internal data structures managed by the smart contracts are not included in the documentation, as they are not directly accessible.

`AFFILIATION` entity

```
pid STRING PK           // provenance id (ref. to blockchain record)
name STRING NOT NULL    // organization name (from blockchain record)
```

`TRADER` entity

```
uid STRING PK           // user id
pid STRING FK NULL      // affiliation (-> AFFILIATION.pid)
cty STRING NOT NULL     // country code (ISO 3166-1 alpha-2)
reg STRING NOT NULL     // registration datetime (ISO 8601 UTC)
```

`ACCOUNT` entity

```
tid STRING PK           // trading account id
uid STRING FK NOT NULL  // owner (-> TRADER.uid)
act INT NOT NULL        // active flag (0|1)
reg STRING NOT NULL     // registration datetime (ISO 8601 UTC)
chd STRING NOT NULL     // last change datetime (ISO 8601 UTC)
chr STRING NOT NULL     // last change by (UID)
```

5 Module Demonstration

This preliminary release of the GOV module supports a basic configuration of the FAME Platform. This configuration is often referred to, within the FAME project, as the *minimum viable product* (MVP): a core set of functionalities enabling dataspace federation, asset publishing and trading. This MVP concept is shared with all the other modules of the Platform, as well as its web-based front end (better known as the FAME Dashboard): extensive coordination work was done, in the scope of task T2.3 Data Marketplace Platform Integration, to ensure consistency and that all cross-dependencies were satisfied. From the GOV module's perspective, this means that this release of the software includes the following subset of operations (see §3 for the complete list and the description of each item):

- Operations related to Trading Accounts:
  - Enroll Account
  - Credit Account
  - Debit Account
  - List Accounts
  - Retrieve Account
  - Retrieve Balance
  - List Traders
  - Retrieve Trader
- Operations related to Federation Members:
  - Onboard Member
  - List Members
  - Retrieve Member
  - Lookup Authority
- Operations related to Users:
  - Invite User
  - Accept Invitation
  - Claim Credential
  - List Users
  - Retrieve User
- Operations related to Support Tickets:
  Support Tickets are not included in the current release
- Operations related to User Requests:
  - Check Request status
  - Enqueue Request
  - Retrieve Request
  - Finalize Request

To summarize, GOV's MVP will allow the FAME Platform to:

- Onboard new members of the FAME federation
- Authenticate users that have been onboarded by any federation member
- Assign special administrative privileges to selected users
- Enable blockchain accounts to trade assets using an internal digital currency for payments
- Enable asynchronous processing of user requests (typically, blockchain transactions initiated from FAME's web front-end)

In the specific context of the MVP demonstration, GOV's server-based software will be hosted on a dedicated runtime environment provided by ENG. The final network address of public and internal service endpoints is not known at the time of writing, as the deployment is still a work in progress.

On the other hand, GOV's smart contracts (like the ones belonging to the P&T and T&M modules) are already deployed on their final environment, which is the FAME Blockchain Infrastructure. This is a permissioned, Ethereum-compatible blockchain network, the nodes of which are currently operated by partners of the FAME consortium.

In the future, all server-based components, from all modules, will be hosted on a production-grade cloud environment that is operated and managed by GFT.

5.1 Deployment of MVP Demonstrator

The MVP demonstrator is a composite system, with several heterogeneous elements seamlessly integrated into a unified working platform. As this deliverable is only focused on the GOV module, we are not going to provide any technical details about the deployment of other platform subsystems. However, the big scheme of things is described in the figure below:

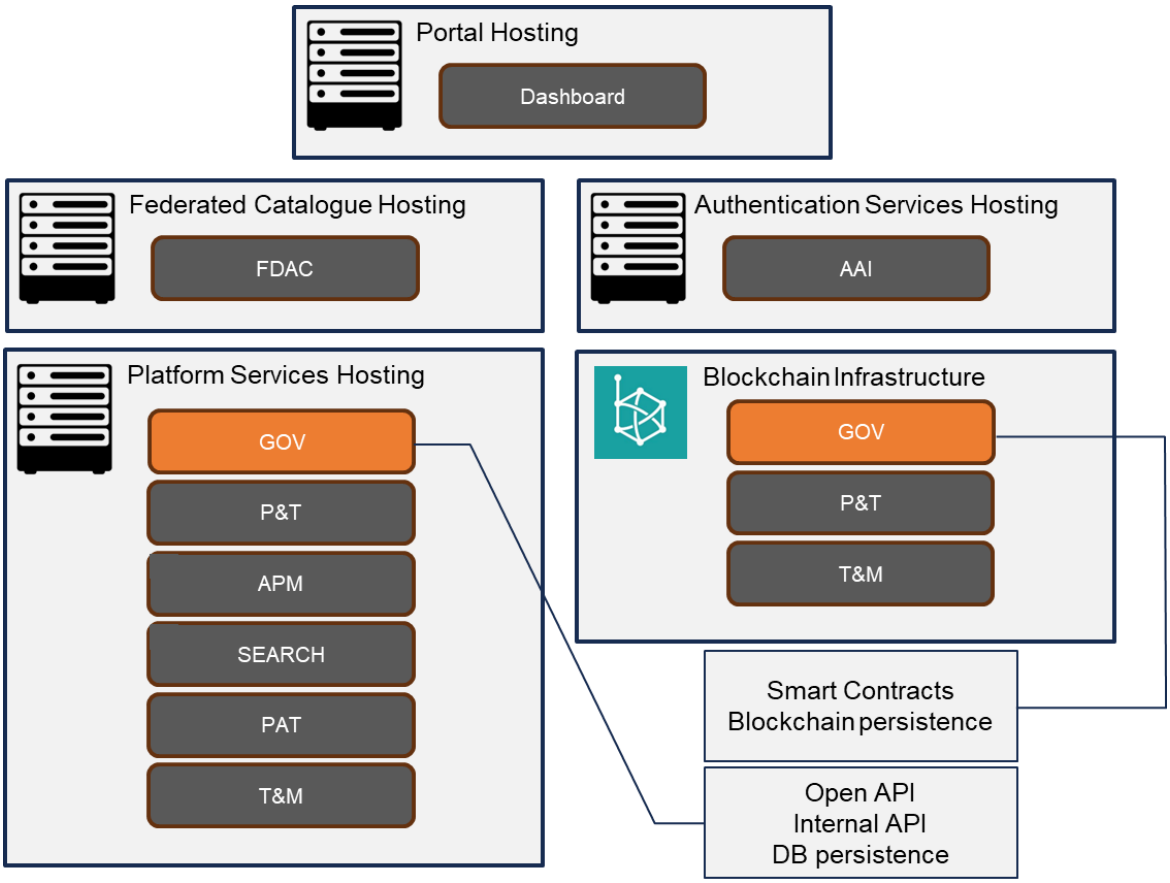

Figure 10 - MVP deployment diagram

As shown above, the GOV module is spread across two different runtime environments: a common Cloud infrastructure (named "Platform Services Hosting" in the figure) and a blockchain network ("Blockchain Infrastructure"). All Open API and Internal API service endpoints, included GOV ones, are hosted on the former. **The Open API service endpoints are exposed over the public Internet, through the secure HTTPS protocol that is configured to run on the 8443 network port**[23]. A

[23] The 443 network port, which is the conventional standard for the HTTPS protocol, is reserved for a future deployment of the Dashboard server in the same hosting environment, thus avoiding potential port conflicts.

screenshot of the Swagger interface[24], showing all GOV-related Open API operations, is provided below:

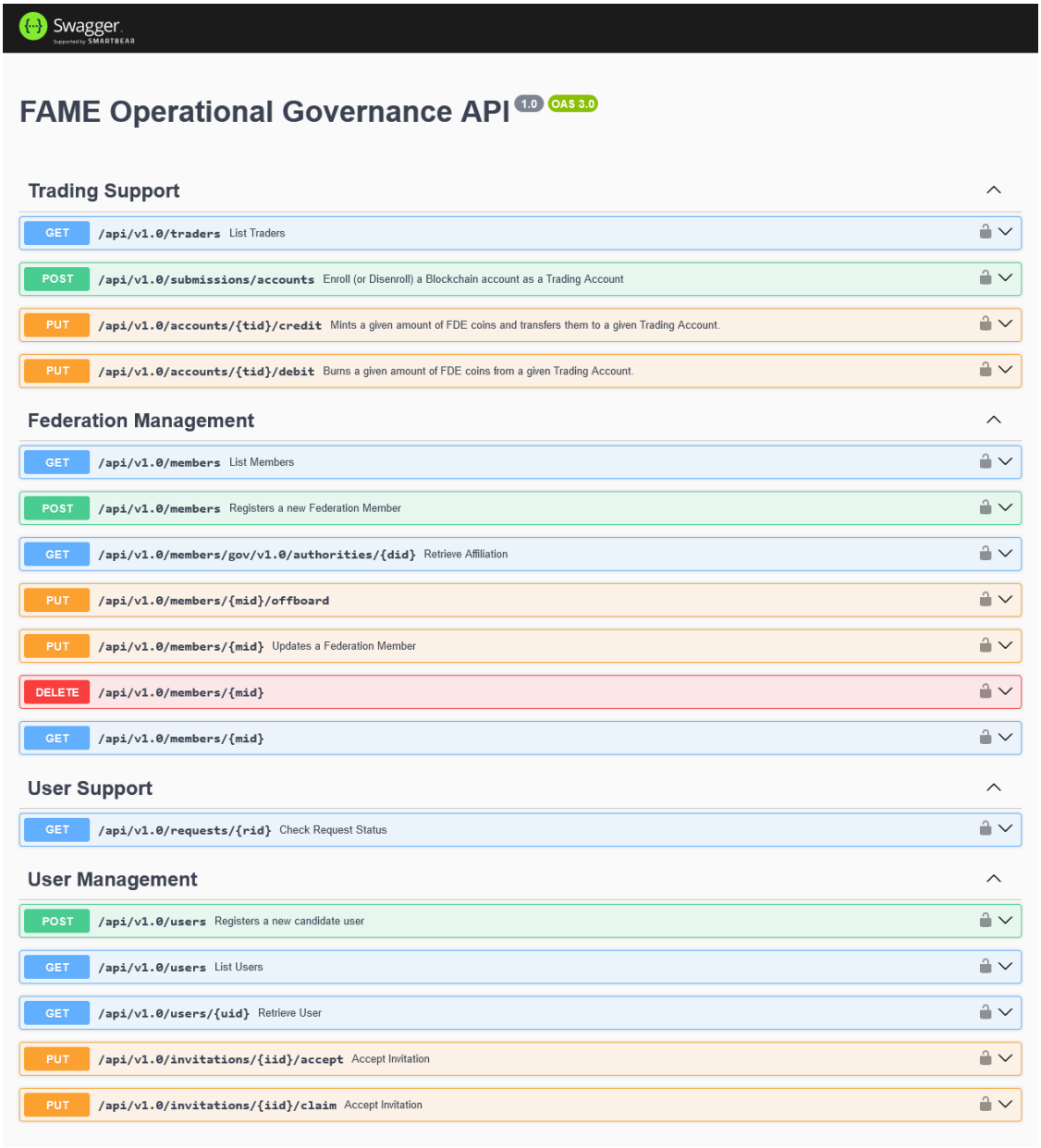

Figure 11 - MVP: deployment of GOV Open API (Swagger interface)

It is worth noting that each element of the Swagger interface can be expanded to display the documentation of that specific operation. An example of this is given below:

[24] See https://swagger.io/

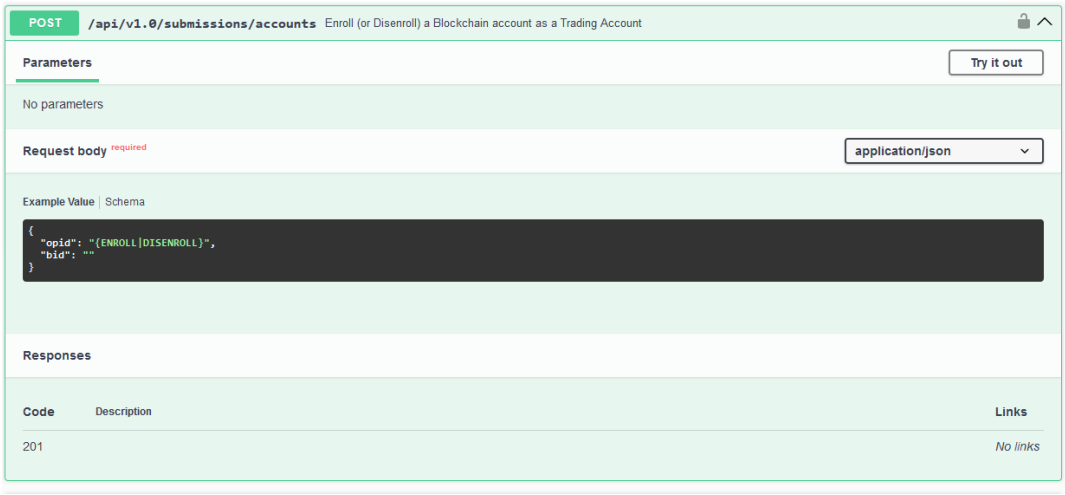

Figure 12 - MVP: example of GOV Open API operation (Swagger interface)

The other runtime environment is FAME's Blockchain Infrastructure, which is one of the outcomes of task T4.1 " Decentralized Data Provenance and Traceability" (see deliverable D4.1 " Blockchain-based Data Provenance Infrastructure", M12 release). This is a Hyperledger Besu v22.10.3 blockchain network, currently consisting of four peer nodes, as can be seen in the picture below. It hosts all the smart contracts of the FAME Platform, including the three that are included in the GOV module (i.e., PaymentToken, FameGovernance, FameBourse - see §4.5.2).

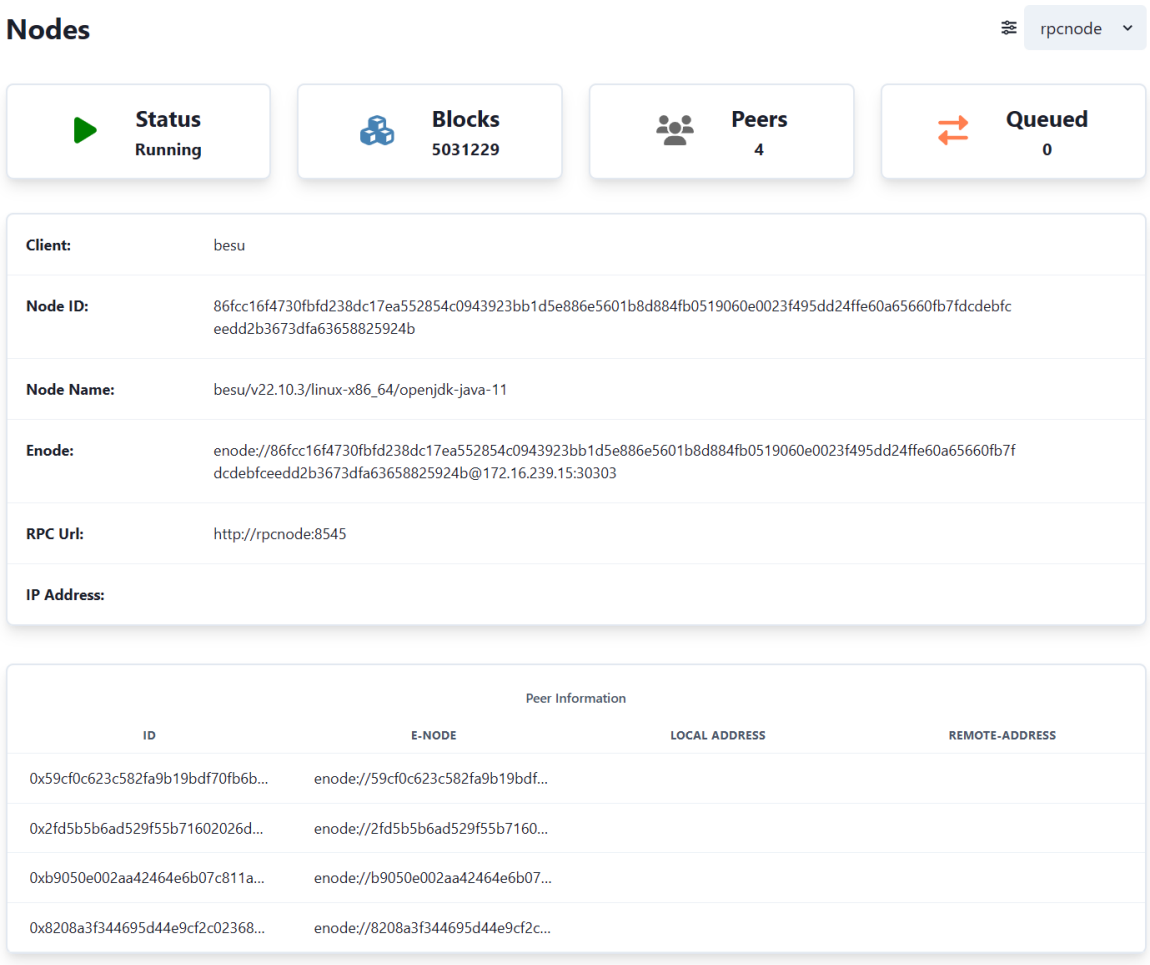

Figure 13 - MVP: deployment of Blockchain Infrastructure

5.2 Custom Deployment of Server-based Software

For deploying your own instance of the GOV module, you can leverage the binary distribution provided by the FAME project. All services are distributed as a set of pre-configured Docker images, and the installation is entirely automated by means of Docker Compose.

5.2.1 Prerequisites and Installation Environment

- Hosting: Linux machine with administrative privileges (recommended OS: Ubuntu 20.04) or Windows machine from version 10.
- Network: connection to the public Internet.
- Required software: Docker and Docker Compose (see https://www.docker.com/)

5.2.2 Installation Guide

You first need access to the source code repository, which is located at this address:

https://gitlab.gftinnovation.eu/fame/fame-framework/gov

Note that access to this repository is restricted: please contact the FAME project's coordination team for getting your personal access credentials.

From the repository, you can download the entire GOV subsystem, including the source code, which is comes under the **Apache 2.0 Open-Source license**. However, in order to install and run the services you only need to retrieve the proper *setup script* for your hosting environment. The script is available from the root of the repository and comes in two flavors: `setup.sh` for Linux systems and `setup.bat` for Windows ones. The script, once run locally on your hosting machine, will automatically download all the required Docker images, configure your Docker containers, and launch the system. This process will look, in your console window, more or less like in the figure below[25]. If the procedure is successful, in the end you will see the *running* label on each line.

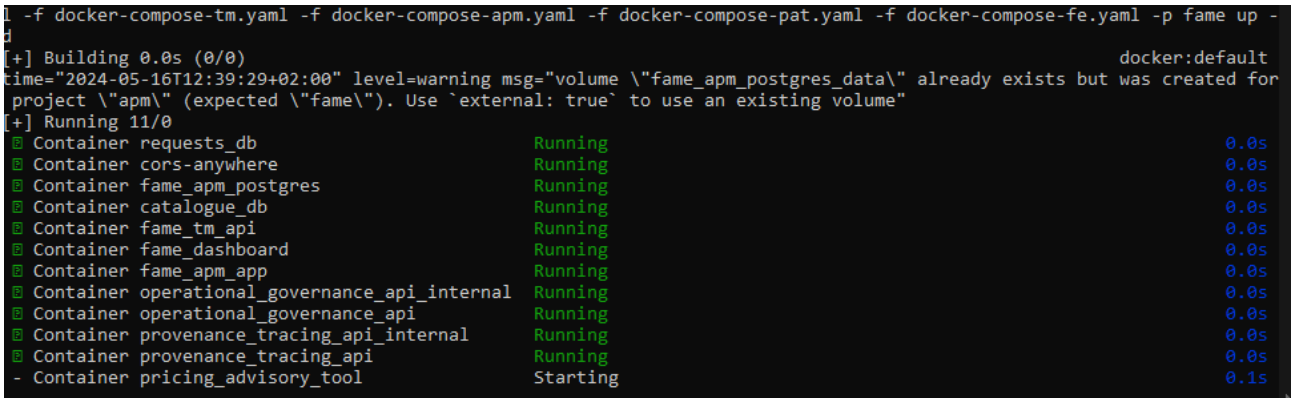

Figure 14 - Installing GOV locally with Docker Compose

The root of the source code repository also contains another couple of scripts: `oid4vci-setup.sh` (Linux) and `oid4vci-setup.bat` (Windows). You only need to run this script if you plan to leverage the User Management component for issuing VOCs to your users. If that is the case, executing the proper script for your hosting environment will cause a new DID to be generated,

[25] Your mileage may vary: the screenshot provided here refers to a different packaging, which includes other FAME modules as well.

configured, and published as the identifier of your Root Authority. The script will ask you for the domain name[26] under which the Open API of your instance is reachable, and will then do two things:

- publish the DID as a `did:web` (see https://w3c-ccg.github.io/did-method-web/), so that the DID Document can be downloaded by anyone from a predefined location that is fully under your control (e.g., https://mydomain.com/.well-known/did.json);
- configure the OID4VCI subsystem to use the DID as the identity of the VOC issuer.

After this process is successfully completed, you should restart all the Docker containers.

5.3 Custom Deployment of Blockchain-based Software

The smart contracts included in the GOV module are distributed as source code: you need to compile them and deploy them on an Ethereum-compatible network.

5.3.1 Prerequisites and Installation Environment

Download and install Metamask wallet from https://metamask.io/ in the browser. In the Metamask wallet, set-up the blockchain network parameters according to your specific environment.

Solidity smart contracts can be compiled using Remix (https://remix-project.org/) or Hardhat (https://hardhat.org/) integrated development environments.

FameGovernance, FameBourse, PaymentToken smart contracts can be deployed on any Ethereum-compatible network. The FAME Blockchain Infrastructure runs the Hyperleger Besu blockchain, which is a *permissioned* Ethereum-compatible platform. Note that, as of now, Hyperledger Besu can support up to the Paris Ethereum Virtual Machine (EVM) versions. Hence, currently, FameGovernance and FameBourse smart contracts can be compiled, and bytecode produced, for example, for Istanbul, Berlin, London and Paris EVM versions will deploy on Hyperledger Besu blockchain but bytecode produced for the newer Shangai EVM version will not deploy. In order to compile smart contracts, integrated development environments such as, for example, Remix or Hardhat, Truffle or VSCode can be used.

For the operation of Governance API that will call smart functions from the server, Ethereum deployment addresseses of FameGovernance and FameBourse are needed. Also, the private key of the owner (private key of the controller Ethereum address) of the FameGovernance contract is needed by the API that will sign and submit governance functions. These addresses and the private key will be kept in an environment file (named .env) and read as environment variables by the web server providing and running the governance API. Since private key is stored on the server, it is of utmost importance that this server be kept secure.

5.3.2 Installation Guide

Smart contract Solidity codes are located in source code repository:

https://gitlab.gftinnovation.eu/fame/**fame**-framework/gov/contracts.

Governance.sol, Bourse.sol and PaymentToken.sol solidity source code files can be loaded to widely used web based Remix or the Hardhat integrated development environments, compiled and deployed to the Fame blockchain. The following specific steps can be followed:

1. Configure the MetaMask wallet so that it points to your blockchain network

[26] Yes, you need one: a plain IP address will *not* work.

2. Compile and deploy FameGovernance, FameBourse, PaymentToken contracts on FAME blockchain by setting evmVersion: Paris in the Remix compilation menu or in hardhat.config.js file in Hardhat. Save the addresses of the deployed contracts.
3. Prepare the following environment variables file .env :

```
# Blockchain provider RPC address and port number
CHAIN_PROVIDER_URL=http://nnn.nnn.nnn.nnn:nnnn
# Ethereum address of deployed Governance contract needs to be put here
GOV_ADDRESS=0xffffffffffffffffffffffffffffffffffffffff
# Ethereum address of Governance contract owner (controller) needs to be put here
GOV_OWNER_ADDRESS=0xffffffffffffffffffffffffffffffffffffffff
# Governance owner (controller) private key needs to be put here
GOV_OWNER_PRIVATE_KEY=0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

4. Using the setFameBourse(..) function of the FameGovernance, set the address of the FameBourse in the governance contract.
5. Using the registerCoinToken(..) function of the FameGovernance, register deployed PaymentToken contract address in the governance contract. Pass the address of the ERC20 PaymentToken contract, 0 as index value (since it is an ERC20 and no token index is used) and the symbol of the token. The symbol for the FAME Digital Euro token is FDE.
6. The governance contract is now ready for the minting and burning of FDE coins using the governance functions mintCoin(..) and burnCoin(..).

6 Conclusions

This deliverable provides a comprehensive overview of the business models and operational governance module for the FAME Marketplace.

We have explored various business models to maximize the value created by the marketplace. An initial list of 18 business models was analyzed and evaluated using a Business Models Selection Matrix, and furtherly enhanced through pilots' validation. The final selection includes subscription-based models, pay-as-you-go (PAYG), pay-as-you-use (PAYU), Data-as-a-Service (DaaS), and freemium models. Such models are intended to maximize the value created by the marketplace and pave the way to exploitation and commercialization.

The Operational Governance (GOV) module platform serves as the gateway for all management and governance functions essential for running an instance of the FAME federation, and is formed by four components:

- Federation Management, providing Open API to administer federation members (e.g., onboarding / offboarding, records management, and membership checks).
- User Management, providing Open API to administer a users' platform pool.
- User Support, an addition compared to initial version, providing Open API to manage ticketing systems and user requests queue.
- Trading Support, providing Open API for trading account permissions, linking to other trading parties, providing insights and historical data, and managing token exchange services.

Such components have been designed, developed and implemented (as described in chapter 4 and 5) to form the minimum viable product (or MVP) of the GOV module, to allow the FAME Platform to:

- Onboard new members into the FAME federation
- Authenticate users who have been onboarded by any federation member
- Grant special administrative privileges to selected users
- Authorize blockchain accounts to trade assets using an internal digital currency for payments
- Facilitate asynchronous processing of user requests (typically, blockchain transactions initiated from FAME's web front-end)

In the next phase of the project, the focus will be on further refining and implementing the selected business models and governance structures. This includes:

- Incorporating User Interaction Metrics: The strategy will integrate interest-based metrics derived from user interactions on the FDAC platform to enhance the pricing advisory mechanism (developed in D4.2) and overall user experience.
- Enhancing MVP Capabilities: As the implementation progresses, additional service endpoints will be included in the GOV module to expand its functionalities. Design and specification is nearly in final stage (but still evolving towards M28), while its implementation is at the beginning – though the MVP expected at M18 will include more.
- Pilot Validation and Feedback: Continuous validation through pilot projects will provide critical feedback, allowing for adjustments and improvements to the business models and operational framework. This iterative process will ensure that the FAME Marketplace remains responsive to user needs and market dynamics.

References

1. Andres, S., Iordanou, C., & Laoutaris, N. (2022). Measuring the Price of Data in Commercial Data Marketplaces. In *ACM Data Economy Workshop.*
2. Azcoitia, S. A., & Laoutaris, N. (2022). A survey of data marketplaces and their business models. *ACM SIGMOD Record*, *51*(3), 18-29.
3. Zhang, M., Beltrán, F., & Liu, J. (2023). A survey of data pricing for data marketplaces. *IEEE Transactions on Big Data.*
4. MiCA, Regulation (EU) 2023/1114 of the European Parliament and of the Council on Markets in Crypto Assets, and amending Regulations, (2023) https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32023R1114 (retrieved 2024-03-25).