NGI ONTO CHAIN

Blockchain for the Next Generation Internet

FINAL FRAMEWORK SPECIFICATION

19/10/2023

Grant Agreement No.: 957338
Call: H2020-ICT-2020-1

Topic: ICT-54-2020
Type of action: RIA

D3.7 FINAL FRAMEWORK SPECIFICATION

WORK PACKAGE	WP 3
TASK	T3.3
DUE DATE	31/08/2023
SUBMISSION DATE	19/10/2023
DELIVERABLE LEAD	IEXEC
VERSION	0.9
AUTHORS	Anthony Simonet-Boulogne (IEXEC) Ambre Toulemonde (IEXEC)
REVIEWERS	Vlado Stankovski (UL) Thanasis Papaioannou (AUEB)
ABSTRACT	This deliverable provides updated specification of i) the ONTOCHAIN framework and architecture compared to the architecture described in the D3.4, ii) its components including those developed by third parties during Open Call 3 and iii) the ONTOCHAIN pilot deployment which will be used to evaluate the project results.
KEYWORDS	Decentralisation, blockchain, trustworthy content, data traceability, trustworthy knowledge exchange, privacy protection, web semantic, service interoperability

Document Revision History

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Version	Date	Description of change	List of contributor(s)
0.1	17/08/2023	Initial draft	Ambre Toulemonde, Anthony Simonet-Boulogne
0.2	25/08/2023	Section 2	Ambre Toulemonde
0.3	28/08/2023	Section 3	Ambre Toulemonde
0.4	29/08/2023	Section 4	Ambre Toulemonde
0.5	30/08/2023	Conclusion & Introduction	Ambre Toulemonde
0.6	11/09/2023	Section 2 & Finalize	Ambre Toulemonde
0.7	15/09/2023	Internal review	Vlado Stankovski, Thanasis Papaioannou
0.8	27/09/2023	Include Thanasis comments	Ambre Toulemonde
0.9	19/10/2023	Include Vlado comments & Finalize	Ambre Toulemonde

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Dissemination Level

Nature of the deliverable:	PU	
PU	Public, fully open, e.g., web	✓
CL	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to ONTOCHAIN project and Commission Services	

DISCLAIMER

The ONTOCHAIN project is funded by the European Unions Horizon 2020 Research and Innovation programme under grant agreement no. 957338.

COPYRIGHT NOTICE

The ONTOCHAIN Consortium is the following:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Participant number	Participant organisation name	Short name	Country
1	EUROPEAN DYNAMICS LUXEMBOURG SA	ED	LU
2	UNIVERZA V LJUBLJANI	UL	SI
3	IEXEC BLOCKCHAIN TECH	IEXEC	FR
4	INTELLISEMANTIC SRL	IS	IT
5	ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS – RESEARCH CENTER	AUEB	EL
6	ELLINOGERMANIKO EMPORIKO & VIOMICHANIKO EPIMELITIRIO	GHCCI	EL
7	F6S NETWORK LIMITED	F6S	IE

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

EXECUTIVE SUMMARY

This document is deliverable "D3.7 Final Framework Specification" of the ONTOCHAIN project funded under the Horizon 2020 Research & Innovation program "ONTOCHAIN– Trusted, traceable and transparent ontological knowledge on blockchain".

The framework specification was produced by Task 3.3, after the execution of ONTOCHAIN Open Call #1 "Research", Open Call #2 "Protocol Suite & Software Ecosystem Foundations" and Open Call #3 "Applications & Experimentation"; it batches the results of the fourteen projects funded and executed between September 2022 and August 2023 and append these outcomes to the infrastructure design provided in the D3.4 deliverable.

The framework specification defines software components that will compose the ONTOCHAIN ecosystem, the application programming interfaces (APIs) that will allow these individual components to communicate, the developed Gateway APIs enabling to have a single entry point to use these individual components.

This document can be serve as guideline for future developers and users willing to use the ONTOCHAIN infrastructure.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

TABLE OF CONTENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

LIST OF FIGURES

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

LIST OF TABLES

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ABBREVIATIONS

API	Application Programming Interface
AWS	Amazon Web Services
DAO	Decentralized Autonomous Organization
DID	Decentralized IDentity
DLT	Distributed Ledger Technology
ERC	Ethereum Request for Comments
EVM	Ethereum Virtual Machine
NGI	Next Generation Internet
OC	Open Call for participation
SDK	Software Development Kit
SSI	Self-Sovereign Identity
VC	Verifiable Credential
W3C	World Wide Web Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

1 INTRODUCTION

This document presents the final version of the ONTOCHAIN framework. It combines the first version of the ONTOCHAIN framework specification initially designed before Open Call 1 (OC1) and the revisions made during the execution of the third party projects funded under Open Call 1, Open Call 2 (OC2) and Open Call 3 (OC3).

The main objective of this deliverable is to provide the final specification of the ONTOCHAIN framework that can be used as a common guidelines and recommendations to future developers and users willing to integrate the ONTOCHAIN services in their own project.

This document offers an update on the D3.5 deliverable, describing the additional features stemming from the achievements of the OC3 projects. It's important to note that a in-depth overview of the functionalities delivered by OC1 and OC2 solutions to the ONTOCHAIN infrastructure is not reiterated in this deliverable, as this information is detailed in the D3.4 and D3.5 deliverables.

The rest of this deliverable is organized as follows:

- Chapter 2, provides the whole ONTOCHAIN architecture based on the D3.5 deliverable with the updated functionalities offered by the OC3 projects. The architecture defines the functionalities and applications realized by all ONTOCHAIN projects, collectively representing the core essence of the ONTOCHAIN vision.
- Chapter 3, presents application programming interfaces for the ONTOCHAIN ecosystem, i.e. the APIs that have been implemented by an OC3 project and that will be presented to application developers and users willing using ONTOCHAIN services. These APIs also defines primitives useful to developers of ONTOCHAIN, e.g. for integrating services and for implementing interoperability between services.
- Chapter 4 describes the update of the pilot use-case, specifically the node deployment process and the current specification of these nodes. There have been no significant changes to the pilot network and the detail can be found in the D3.4 and D3.5 deliverables.
- Chapter 5 provides concluding remarks for this deliverable.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

2 ONTOCHAIN ARCHITECTURE

The ONTOCHAIN software ecosystem consists of a novel protocol suite grouped into high- level application protocols, such as data provenance, reputation models, decentralised oracles, market mechanisms, ontology representation and management, privacy aware and secure data exchange, multi-source identity verification, value sharing and incentives and similar, and core protocols that include smart contracts, authorisation, certification, event gateways, identity management and identification, secure and privacy aware decentralised storage, data semantics and semantic linking.

2.1 ARCHITECTURE DESIGN

Since the D3.5 deliverable, the architecture design has remained largely unchanged. Indeed, a modular approach has been implemented to ensure scalability, openness and high performance. The ONTOCHAIN ecosystem is structured with four layers as depicted in Figure 1: APPLICATIONS, ONTOLOGIES and DISTRIBUTED LEDGER on the left side that each builds on the functionalities offered by the lower layer, and INTEROPERABILITY MODULES AND PROTOCOLS that is a cross-layer focusing on interfaces and interoperability.

As a final result, Figure 1 also illustrates the projects actively contributing to the development of modules:

- OC1 for building the foundations of the ONTOCHAIN ecosystem.
- OC2 integrated the outcomes of OC1 and implemented the ONTOCHAIN infrastructure.
- OC3 completed the missing building block and exploit the ONTOCHAIN infrastructure in real-life use cases.

Some of the modules in these layers have been modified to take into account the progress made during the execution of OC2 and OC3.

2.1.1 INTEROPERABILITY MODULES AND PROTOCOLS LAYER

Core protocols The Identity Management and ONTOCHAIN API Gateways modules have not be modified and aim to deliver the same objective as described in the D3.5 deliverable.

The Authorization and Certification modules have been merged to develop SSI solu-

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

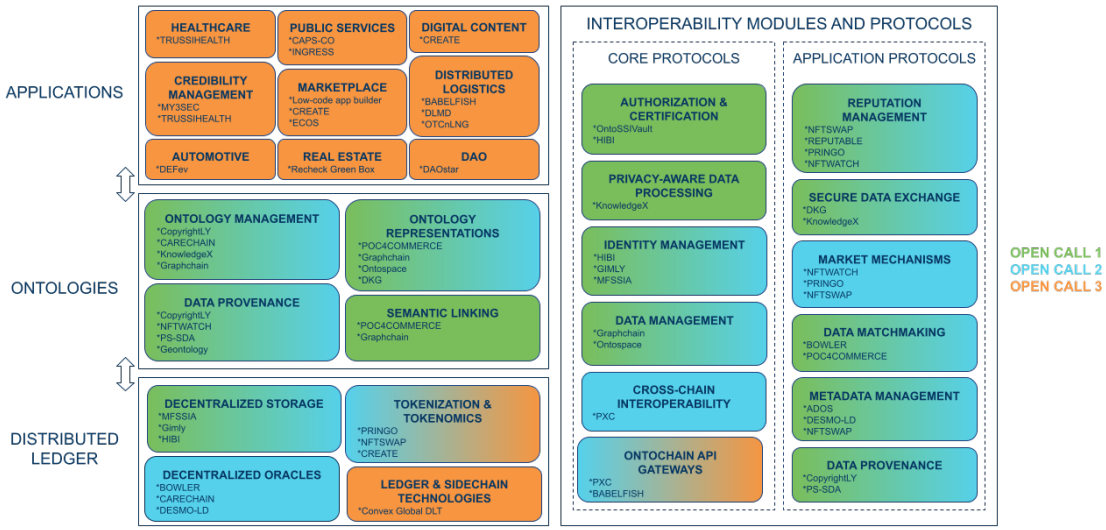

FIGURE 1: ONTOCHAIN ARCHITECTURE AND COMPONENTS DIAGRAM.

tions that ensure compliant data with relevant standard (e.g. W3C DID) and regulation (e.g. GDPR).

The module previously named 'Crosschain Transactions' has been renamed to 'Cross-chain Interoperability' to provide a more precise description of its unchanged objective as defined in the D3.5 deliverable: facilitating the interoperability between two relatively independent Blockchains.

The Decentralized Storage module has been removed, as it is more relevant and appropriately placed within the DISTRIBUTED LEDGER layer.

The additional two modules are the following ones:

- Privacy-aware Data Processing: this focuses on ensuring data privacy and security. It allows for the processing of sensitive data while preserving user privacy.
- Data Management: this is responsible for efficiently storing, retrieving, and organizing data. It enables the secure and decentralized storage of data.

Application protocols The Secure data exchange and Data Provenance modules have not be changed.

The modules previously named 'Reputation Models' and 'Semantic Match-making have been renamed to 'Reputation Management' and 'Data Matchmaking' respectively. Their objectives remain unchanged as detailed in the D3.5 deliverable.

The Multi-source Identity Verification module has been removed, as the projects com-

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

pleting the Authorization and Certification module have successfully achieved the objective of registering and verifying individual digital identities of physical objects via trusted data from multiple sources.

The Decentralized Autonomous Organization has been developped as an application use-case by a OC3 third party and thus placed in the APPLICATION layer.

Thus, these two additional modules have been added to fulfill the essential services required for this layer:

- Market Mechanism: this provides the functionalities for decentralized markets and trading platforms. It facilitates the creation of smart contracts and tokens, allowing for peer-to-peer trading, auctions, and various market mechanisms.
- Metadata Management: this involves the organization and retrieval of additional information about transactions, assets, or records. It enhances the usability and searchability of data.

2.1.2 DISTRIBUTED LEDGER LAYER

This layer remains largely unchanged. The module previousouly named 'Digital Currency Tokenomics' has been changed to 'Tokenization & Tokenomics' to be more precise about it unchanged objective as defined in the D3.5 deliverable. The name and objective of the other modules are the same as defined the D3.5 deliverable.

2.1.3 ONTOLOGIES LAYER

The Data Provenance, Semantic Linking and Ontology Representations modules remain unchanged. The module previously named 'Semantic Matchmaking And Reasoning' has been renamed to 'Ontology Management' in order to comprise more functionalities than finding connection between items. Indeed, the module includes the the following ones:

- Ontology Management: this concerns the creation, organization, and maintenance of ontologies, which are structured representations of knowledge or domain-specific concepts and their relationships. The primary purpose is to define a shared vocabulary and a formal model for a specific domain.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

2.1.4 APPLICATION Layer

This layer was the focus of OC3 and has been extended by the projects funded during this open call. Thus, 9 modules have been defined and successfully implemented by these OC3 solutions:

- Healthcare: this aims to securely store and manage patient records, ensuring data integrity, privacy, and interoperability among different healthcare providers. It can also facilitate drug traceability, clinical trials, and health research.
- Public Services: this helps to facilitate and enhance transparency and efficiency in both personal and industrial operations.
- Digital Content: this focuses on protecting intellectual property rights and manage digital content distribution. It provides content creators with a secure and transparent way to manage copyrights, royalties, and content provenance.
- Credibility Management: this helps to verify and establish trust in various contexts, such as certifications, to prevent fraud and to improve the credibility of online information.
- Marketplace: this enables secure peer-to-peer transactions, reducing the need for intermediaries. They are used in areas like e-commerce, supply chain management, and decentralized finance (DeFi).
- Distributed Logistic: this enhances supply chain management to track the movement of goods while improving traceability, reducing fraud, and ensuring the authenticity of products.
- Automotive: this helps to record and manage vehicles history in order to improve their maintenance.
- Real Estate: this simplifies aggregation, normalization and secures online and offline data storage of real estate data.
- Decentralized Autonomous Organization (DAO): this module can run a Blockchain protocol entirely and autonomously with the help of smart contracts. Thus, it circumvents the need for human intervention or centralized coordination and helps to build a trustless system.

2.2 COMPONENTS DESCRIPTION

The system architecture described in the previous section has been actualized through the integration of software components developed by participants selected during the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

three ONTOCHAIN open calls.

Over the course of the last year, the OC3 third parties participated to complete not only their initial layer, APPLICATIONS, but also two other layers, DISTRIBUTED LEDGER and INTEROPERABILITY MODULES AND PROTOCOLS. This section introduces these new components of the ONTOCHAIN infrastructure provided by the OC3 projects.

2.2.1 Babelfish

Description BABELFISH proposes to describe services on a technical, semantic, and governance layer and will implement a component that uses such descriptions to translate interfaces (APIs), data, and data agreements from a foreign (and maybe proprietary format) to an interoperable format understood by the recipient. A registry maintains a list of all services and thus spans up an interoperable data space.

Application Programming Interfaces BABELFISH completes the ONTOCHAIN API Gateways module by implementing Gateway API for developers and users of ONTOCHAIN services, already introduced in Section 3 of the D3.5 deliverable. The detail of this APIs is provided in Section 3. Three additional building blocks have been developed to provide necessary functions for applications in the ONTOCHAIN environment and beyond:

- *Identity Management:* provide identities for organisations, users, and datasets using decentralised identifiers (DIDs) and use Verifiable Credentials for attestation.

The `did:oyd` method[1] is an example of a non-blockchain based DID method and provides a self-sustained environment for managing digital identifiers: it cryptographically links the identifier to the DID Document and uses linked provenance data in a public log to ensure resolution to the latest valid version of the DID Document.

Information about DIDs, DID Documents and associated logs are stored in an OYDID repository. This repository is a centralised but can be duplicated to other repositories, providing decentralization. Anyone can host a repository and it is up to the DID owner to select a trusted provider. At any time, the `did:oyd` method's decentralization allows at any time seamless transition to other repositories.

The following resources documents the improvements of the `did:oyd` DID method:

- updates and improvements as commits on the OYDID Github repository: https://github.com/OwnYourData/oydid
- implementation and documentation of DID Delegation in the following blog post that was picked up on Twitter and generated some interest: https://www.

[1] W3C conform DID method specification: https://ownyourdata.github.io/oydid/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ownyourdata.eu/en/did-delegation/

- did:oyd is now the only DID method that covers 100% of the W3C DID Test Suite: https://github.com/w3c/did-test-suite/pull/219
- as part of the compliance work we developed the online DID Lint Service: https://didlint.ownyourdata.eu

The benefits of did:oyd DID method are the following ones: a lightweight and standard-conform implementation, a maximum of privacy with local deployment and a Native integration with Semantic Container storage provider.

- Data Agreements: legally binding contracts that stipulate the terms and conditions for sharing, accessing, and using data between two or more parties to ensure transparency and mutual consent in the data exchange process

Data agreements set the usage policies for data access, ensuring that all parties involved adhere to the agreed-upon rules and regulations. To facilitate data sharing in a more structured and well defined way a vocabulary or ontology, a Domain Specific Data Agreements (D2A) is used. Such agreements allow organisations to communicate through an intermediary the allowed usage policies for data access within a specific domain. The intermediary has then all the information to match and validate the usage policy in order to create a Domain Specific Data Disclosure Agreement (D3A).

One of the key technologies enabling data agreements are Verifiable Credentials (VCs). VCs are digital representations of information that a verifier might require, such as the age or the citizenship of an individual. In the context of data agreements, a VC could be created to represent the agreement itself. This VC would contain the details of the agreement and could be cryptographically signed by the parties involved. This ensures the authenticity and integrity of the agreement, providing a secure foundation for data exchange.

- Model Management: describe data structures and use the information for data capture (with HTML forms), validation (based on SHACL1), and transformation

The Semantic Overlay Architecture (SOyA) is a lightweight, semantic-web based approach to describe data structures in simple terminology[2]. This description includes groups of data records with the same attributes, references between data records, and additional information in the form of overlays for these data structures.

At the core of the SOyA approach is the SOyA structure, a YAML-based data model for describing graph data, which consists of one or more soya:Base, that represent RDF classes and their properties, and zero or more soya:Overlay, that provides additional information and context to soya:Base as well as processing definitions. Furthermore, to

[2] W3C conform specification: https://ownyourdata.github.io/soya/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

support developers in conducting the most common data processing for graph data, a number of predefined `soya:Overlay` have been defined such as `soya:AnnotationOverlay` for data model description in human readable terms and `soya:ValidationOverlay` for constraint checking.

It is important to note that SOyA has the same flexibility as RDF (Resource Description Framework) to describe data structures, i.e., any kind of data or documents can be described. With SOyA it is possible to register any current and future data models handled by ONTOCHAIN services and applications.

The improvements of SOyA are documented in the following resources:

- updates and improvements as commits on the SOyA Github repository: https://github.com/OwnYourData/soya
- specifically a new function for the automated creation of transformation overlays based on alignment information was created and demonstrated in the following tutorial: https://github.com/OwnYourData/dc-babelfish/tree/main/tutorial/5_Transformations
- using the functionality of validation overlays the DID Lint Service was made possible and was eventually integrated into the stable version of the UniResolver at the Decentralised Identity Foundation: https://resolver.identity.foundation/

2.2.2 Convex Global DLT

Description Convex provides an energy efficient and scalable decentralised ledger Technology (DLT) on a permissionless public network. It has been designed as a substrate for decentralised economic transactions and smart contracts for real-time value exchange. It supplies documentation, ecosystem services, training and API implementation support for selected ONTOCHAIN use-case partners. The network operates with any number of peer operators that confirm the global state in real time at thousands of transactions per second.

Application Programming Interfaces Convex Global DLT enables to strengthen the decentralized aspect of the ONTOCHAIN ecosystem and add a new DLT into the Distributed Ledger module. The following API for SDK modules have been implemented:

- Convex Shell

Initially a background component, Convex Shell is an application for running Convex Lisp in the terminal. At the core, Convex Lisp is their Smart Contract programming language. This tool extends the Convex Virtual Machine (execution engine) to turn this

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

language into a scripting language offering all the features commonly needed for development and network operations. Their core technology being written in Java, this tool allows Convex developers to indirectly access this tooling without any JVM experience.

Most of the other modules described in this section will be presented as extensions to Convex Shell, requiring only knowledge of Convex Lisp, meaning they maximise access and reusability of those deliverables with respect to users having an interest in Convex and Convex Lisp.

- STDIO Interface - Convex Shell extension

Integration of their tooling with other environments and languages has been a recurring topic when discussing with other ONTOCHAIN projects. While Convex Shell can be used as a standalone application, this module offers a simple way of accessing it from any programming language by using conventional inter-process communication.

- Unit Testing Framework - Convex Shell extension

Testing smart contracts often involves a subpar experience, such as testing Solidity contracts from Javascript. In contrast, this module offers a simple unit testing library so that tests can be fully written in Convex Lisp, without involving any third-party layer, minimising context switching and maximising code reuse. It also used to test modules from this project written in Convex Lisp.

- Etch Stress Testing - Convex Shell library

Etch is the companion database for the Convex Virtual Machine, optimised for Convex data. This is what peer nodes use to persist the global state of the network as well as blocks of transactions and related data. Convex Shell has full support for Convex data generators, which can be easily composed for generating any data in a random but controlled way. This module builds on those data generators and offers a framework for studying the performance of Etch by generating random data and measuring write speed. It also performs an integrity check, ensuring that the persisted data can be read back.

- Peer Deployer - Convex Shell library

Originally, there were already several ways for starting a Convex peer node for processing network transactions. For instance, see:

1. https://convex-dev.github.io/convex/convex-cli
2. https://convex.world/cvm/peer-operations

This module offers functions automating those steps when setting up networks, greatly

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

reducing the complexity via 2 simple functions run with the Convex Shell.

- Simulation Scenarios - Convex Shell library

Preparing transactions, such as for the purpose of benchmarking or simulation, can be repetitive and tedious to do without introducing bias. Building on the Convex data generators, this module offers a straightforward abstraction for generating random transactions that follow a well-known scenario, in a fully reproducible way. This module is an important prerequisite for simulating realistic scenarios.

The first set of scenarios aims to test common constructs that either have comparable implementations on other mainstream platforms or could be written from scratch in a straightforward way. The second set of scenarios aims to represent Convex-specific features, either unique to Convex or not commonly available on other L1 platforms. Given how our generative tooling operates, all those scenarios are fully reproducible. Given the same set of parameters for a scenario, the exact same transactions will be generated for those parameters.

- Client Load Generator - Convex Shell library

Since Convex Shell offers access to our binary client, used for transacting over Convex networks, this module provides a Client Load Generator able to connect to a running network and generate transactions according to Simulation Scenarios.

The Client Load Generator can be bucketed as to run several instances against a network. This will spread load generation against all Client Load Generator instances that will typically run on different machines. This avoids biasing results in case of Client Load Generators struggling to generate the appropriate load due to the technical limitations of a single machine.

- Local Simulation Runner - Convex Shell library

In order to test the tooling exposed above and validate the design before extensive cloud simulation, this module provides a straightforward way for starting a local network on a single machine according to a scenario and running a load against it.

This module does not collect any other metrics beyond the finality results outputted by the Client Load Generator. Its primary goal is to ensure that all the deliverables required for cloud simulations can indeed fit together and behave as necessary.

- AWS Simulation Orchestrator - JVM library written in Clojure

Culmination of this project, this module is a complex combination of previous modules that fully automates the deployment of a test network across data centres using AWS CloudFormation, via a single function managing the whole process of:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- Deploying EC2 instances for peers, using the Peer Deployer, and bootstrapping the network
- Deploying EC2 instance for Client Load Generators
- Running a Simulation Scenario
- Collecting all metrics
- Shutting down the test network automatically after a given interval of time

Collected metrics and data, common to all scenarios, are:

- Logs of all peer servers
- Logs of all Client Load Generators
- Finality statistics (delta between sending a transaction and getting a result confirmed by consensus), in milliseconds:
    - Average
    - Standard deviation
    - Quartiles
- Network statistics based on network data retrieved from an Etch instance:
    - Disk usage (size of the instance file)
    - Number of blocks
    - Consensus information specific to Convex: Proposal Point and Consensus Point
    - Number of transactions confirmed by consensus
    - Block size quartiles (number of transactions per block)
    - Average Blocks per Second
    - Average Transactions per Second
- Machine statistics for peer instances, according to metrics retrieved from AWS CloudWatch:
    - Quartiles of CPU utilisation (percent)
    - Quartiles of memory usage (MB)
    - Total volume of inbound network data (GB)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- Total volume of outbound network data (GB)
- Average inbound network speed per peer (MB/s)
- Average outbound network speed per peer (MB/s)

This module can also interface with peers deployed externally. This will be the case for their lab experiments which will consist of running such a test network as well as a physical machine for which we can measure energy consumption.

2.2.3 DLMD

Description The Decentralized Last-Mile Delivery (DLMD) solution revolutionizes the last-mile delivery ecosystem by integrating blockchain technology with a user-centered application. The solution is designed to enable efficient, transparent, and secure parcel transactions, utilizing parked vehicles as delivery points. This innovative approach addresses the challenges of conventional delivery systems, offering improved service for delivery companies, vehicle owners, and parcel senders and receivers. Through its unique NFT-based parcel identification and vehicle access system, DLMD ensures the seamless transfer of parcels while optimizing delivery times and costs.

Application Programming Interfaces DLMD provides specification for the SDK and the smart contract, composed of the following modules.

Module *Authentication* The module provides authentication and authorization using DID, by rewriting a part functionality of the Gimly module.

`createEncryptedWallet(pin)` Creates an encrypted wallet and stores its data in secure storage, the user is then prompted to save the mnemonic phrase offline.

`createDid()` Creates DID and stores it in secure storage.

`getUserDid()` Gets the users DID.

`sign(message)` Signs a message using the users wallet.

Module *Login and registration* This module provides the ability to register and login users. When registering or logging in with a wallet, users need to call /wallet-auth-message endpoint of this module in order to obtain data required for signature and timestamp arguments. Endpoints: `https://<base_uri>/auth/`

POST `/register/wallet` Creates a new user account and returns login data.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

POST `/login/wallet` Logs an existing user in.

GET `/wallet-auth-msg)` Provides a message for the user to sign, as well as a timestamp parameters needed to perform a sign in.

Module *User* This module handles user data. Endpoints: `https://<base_uri>/users`.

GET `/me` Returns data associated with currently logged in user.

PUT `/update` Updates data associated with currently logged in user, and returns updated object.

GET `/details` Returns detail data associated with currently logged in user.

POST `/details` Updates and returns detail data associated with currently logged in user.

GET `/registered-wallets` Registered wallet addresses. Used to limit the mint ability on the SC.

Module *Box* Handles creation and management of boxes. Endpoints: `https://<base_uri>/box`.

POST `/create` Creates a box and returns its data as a json object.

PATCH `/<id>/update` Updates the box and returns its updated data as a json object.

GET `/<id>/permission` Can user access the box.

GET `/` Lists boxes that belong to the currently logged in user.

GET `/data` Data about the specific box.

POST `/connect` Permission data after the box has been connected.

Module *Permission system* This module controls whether a user is allowed to open the box or not. Endpoints: `https://<base_uri>/box`.

GET `/box/:id/access-key` Generates and returns an access key for the box, based on the challenge provided.

POST `/box/:id/permission/set` Grants the user the given permission for the given box.

POST `/box/:id/permission/revoke` Revokes the user the given permission for the given box.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Module *Reputation system* This module handles rating of courier users and boxes. Upon receiving a new rating, this module also recalculates users or box average rating and updates the cached rating on the User or Box model accordingly. Endpoints: `https://<base_uri>/reputation`.

POST `rate-transaction` Creates a rating of the interaction. For each parcel, two ratings are necessary one for the courier, and one for the box.

GET `/average` Calculates average rating for a given recipient, based on rating type and recipients ID.

GET `/:wallet` Calculates average rating for a given recipient, based on wallet address.

Module *Parcel transaction* This module handles parcel creation and later transactions, such as depositing the parcel into a box, or withdrawing the parcel from a box. Endpoints: `https://<base_uri>/parcel`.

GET `/` Lists parcels matching the filter.

GET `/:id` Returns data for a given parcel.

POST `/create` Creates a new parcel based on user and box IDs, and returns it as a json object.

POST `/create/by-wallet` Creates a new parcel based on wallet addresses, and returns it as a json object.

PATCH `/update/:id` Updates a parcel and returns updated data as an object. Where identifiers are necessary, this request uses database IDs for this purpose.

PATCH `update/by-wallet/:nftId` Updates a parcel and returns updated data as an object. Where identifiers are necessary, this request uses database IDs for this purpose. Location can also be updated through location API.

POST `/:id/deposit` If conditions for parcel deposit are met, it marks the parcel as deposited, revokes couriers access to the box, and grants the package recipient the access to the box. If conditions for deposits are not met, an appropriate error is returned.

POST `/:id/withdraw` If conditions for parcel withdrawal are met, it marks the parcel as withdrawn and revokes the recipient's access to the box. If conditions for deposits are not met, an appropriate error is returned.

GET `/:id/location` Returns parcels location.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Module *Location* Location API creates and updates locations. Endpoints: `https://<base_uri>/location`.

POST `/create` Creates a new location and returns the created location.

POST `/:id/update` Updates an existing location and returns updated data.

POST `/box/:boxId/precise` Creates a precise location for the given box.

PATCH `/box/:boxId/precise` Updates box precise location and returns updated data.

GET `/box/:boxId/precise` Gets locations precise location.

Module *Parcel NFT smart contract (ERC-721)* Parcel NFT smart contract extend the standard NFT SC by storing the following data: wallet address of the current owner of the parcel and handle (link) to the data providing encrypted metadata (unique for each transaction). Extended and additional functions are the following ones.

`mint` Provides mint - NTF construction functionality.

`eligibleToWrite` Checks of the user has permissions to write to the SC.

`transferOwnership` Transfers the NFT ownership.

`burn` Remove the existence of the NFT.

`getTransactionIndex` Unique transaction index.

Module *Reputation system smart contract* It will enable reputation management for the users.

`evaluateUser` Provides functionality to evaluate the user.

`getScore` Returns the score of the user.

2.2.4 Recheck Green Box

Description The ReCheck Green Box is a digital building logbook that aggregates, normalizes and secures online and offline data about buildings. The solution aggregates different types of documents, certificates, etc, linked to the life cycle of a building starting with design plans, execution plans and reports and maintenance protocols. The data is stored in a semantic data lake for further usage and querying. The data origin,

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

authenticity and its properties are protected by blockchain records.

Application Programming Interfaces Recheck Green Box provide the API specification for REST Services, divided into the two following modules:

Module *Remote Module Monitoring (RMM)* The API service is used by remote installed modules and Aggregator service. Modules are preconfigured to send their sensing data to that api service on a regular basis.

POST `/{token}` Identify provider, manufacturer and model. If they are valid the service will store the sensing data.
GET `/lastData` last sensing data for each stored module

Module *Graph Query API* The Graph Query API is designed to serve stakeholders to integrate the logbook for the purpose of direct data extraction and as a connection data source for business intelligence reporting. **POST** `/asset/create` Used by the aggregator and authorised services / users to push to DKG new data or update existing data of the digital asset with assertions.
POST `/graph/query` Authorised users (stakeholders) provide SPARQL query to get the desired information as RDF statements for BI and reporting.
GET `/graph/query/organization/measurement` Authorised users (stakeholders) get information for organization building devices with measurements for specific time period.

2.2.5 My3Sec

Description My3Sec is a comprehensive solution designed to enhance the efficiency and transparency of remote work processes. Developed with a focus on the use cases of proving skills, seeking good candidates, and tracking workers' growth for a project, My3Sec is a fully decentralized, transparent, and efficient system for tracking people's career growth and improving employers' project management through a verified skill-oriented approach. With its user-friendly Web3 frontend, robust contract architecture, and democratic governance system, My3Sec is poised to revolutionize the way organizations and individuals approach skill verification, candidate search, and career progression.

Application Programming Interfaces My3Sec provides the different modules for My3SecHub Smart Contract. It manages user profiles and organizations, interacting

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

with other contracts like My3SecProfiles, EnergyWallet, TimeWallet, and Organization. The modules for My3SecHub Smart Contract are given in the following parts.

Module *Contract Setup* These functions are used for setting up the My3SecHub contract by linking it to the necessary contracts for profiles and wallets.

`setMy3SecProfilesContract(address contractAddress)` Link the My3SecProfiles contract with the My3SecHub contract. Only the contract owner can execute this function.

`setEnergyWalletContract(address contractAddress)` Link the EnergyWallet contract with the My3SecHub contract. Only the contract owner can execute this function.

`setTimeWalletContract(address contractAddress)` Link the TimeWallet contract with the My3SecHub contract. Only the contract owner can execute this function.

Module *Profile Management* These functions provide an interface to interact with profiles and the associated energy tokens.

`getDefaultProfile(address account)` Returns the default profile of the given user.

`getProfile(uint256 profileId)` Return the profile with the given profile ID.

`setDefaultProfile(uint256 profileId)` Set the given profile as the default profile for the sender.

`createProfile(DataTypes.CreateProfile calldata args)` Create a new profile with the given metadata URI and assigns it an initial energy balance.

`giveEnergyTo(uint256 profileId, uint256 amount)` Transfer energy from the sender's default profile to the given profile.

`removeEnergyFrom(uint256 profileId, uint256 amount)` Remove energy from the specified profile and gives it to the sender's default profile.

Module *Organization Management* These functions provide an interface for managing organizations.

`getOrganizationCount()` Return the total number of registered organizations.

`getOrganization(uint256 index)` Retrieve the address of the organization at the given index. Reverts if the index is out of bounds.

`createOrganization(string calldata metadataURI)` Create a new organization with the given metadata URI and assigns ownership to the sender.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

`registerOrganization(address organizationAddress)` Register an existing organization in the contract. Reverts if the organization is already registered, if the address doesn't point to a contract, or if the contract doesn't comply with the IOrganization interface.

`joinOrganization(address organizationAddress)` Let a user join an existing organization. The sender's default profile will be used.

`leaveOrganization(address organizationAddress)` Let a user leave an organization. The sender's default profile will be used.

`logTime(address organizationAddress, uint256 projectId, uint256 taskId, uint256 time)` logs time spent by a user on a specific task in a project. It uses the sender's default profile, removes the specified time from the user's time wallet, and updates the task's total time in the organization.

2.2.6 ecOS

Description ECOS is the full stack platform for the Energy Community, the platform enables a transparent, accountable system capable of creating economic value through Token Model for all the users involved in the energy community.

Application Programming Interfaces ecOS is built on top of their existing IoT layer. They provide complete API definitions in two distinct set of APIS:

- The IoT Layer API: https://documentation.apio.network/api
- The ecOS API: https://app.swaggerhub.com/apis-docs/fatmatto/ECOS/1

In the following paragraph we include just the ecOS API that are composed by Energy Data Module and Blockchain Module.

Module *Energy Data and Blockchain* In the Blockchain module, only transactions essential to the operation of the ECOS infrastructure are included. For all GET type functions related to the contract (such as retrieving the Token Balance of an account), ethers.js is used directly by loading the contract's ABI (Application Binary Interface). For all other more generic functions related to the Blockchain (such as the list of transactions), the capabilities of ethers.js are utilized. This approach ensures a streamlined, secure and effective way of interacting with the underlying blockchain.

POST `/projects/PROJECT_ID/communities` Creates a new community. This operation requires admin privileges.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

POST `/projects/PROJECT_ID/users/register` Register a new user to the platform. Not an admin, but a regular end user of ecos, able to use the mobile app. The endpoint will start a registration flow.

POST `/projects/PROJECT_ID/users/authenticate` Authenticates the end user.

POST `/projects/PROJECT_ID/wallet/init` Initializes the wallet of the end user. The endpoints requires that the user provides parameters for the wallet initializations, and the address. This will link the user on the platform to the wallet.

GET `/projects/PROJECT_ID/communities/COMMUNITY_ID/overview` Returns general information about the community and about the contribution of the current user to the community and its consumption data.

GET `/projects/PROJECT_ID/communities/COMMUNITY_ID/analysis` Goes deeper into the details of energy consumption generated by the current user, returning a drill down of the consumption from grid, storage and the savings.

GET `/projects/PROJECT_ID/incentive/set` Sets the specified incentive for a given user. This operation requires admin privileges.

2.2.7 DEFev

Description DEFev project, with its blockchain-based solution, resolves interoperability and actor class complexity issues by providing a common infrastructure that integrates different parties into one unified ecosystem. The charging stations, through their IoT server, have a digital twin in the blockchain. This allows to manage a micro charging operator as easily as a user, opening the path to seamless peer-to-peer (P2P) charging. Furthermore, the on-chain data management ensures traceability of transactions in a manner that ensures trustworthiness and transparency and grows the reputation of each entity.

Application Programming Interfaces DEFev provides the following modules:

Module *DAPP DEFev*

`/identified/select-evse/map-selector` Displays the map with the location of the user and all the valid EVSE as markers.

`/identified/wallet-management` Displays the tokens of the user (no use of coins in Bellecour).

`/identified/transaction-history` Displays the history of payments(charges).

`Choose EVSE` - - Scan the QR Code or - Go to the map page.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

`Detail EVSE` Displays the information page of the EVSE. Has a button to show the comment page.

`Charge EVSE` Asks for duration and displays the price of the transaction. If ok, submits the charge request to the Pilot. Once the charge is on, we can interrupt it with a button. Once the charge is finished, we can post a score in the Reputation.

`my Account` Displays the address currently connected and the settings the user defined.

`OC1 Price Oracle Smart Contract` Displays the current price index and its description. Ex: Prices are in EUR per kWh. This helps the member define his margin. FR1 : 0,14; Regulated Price per kWh for individual in France. FR2 : 0,28; Regulated Price per kWh for Corp in France. FR3 : 0,37; Spot price in France.

Module *Content Server* This is the public IPFS service, or any other compatible storage service. It is only read by the Dapp when displaying information about the EVSE.

Module *EVSE Address Book (Main Smart Contract)*

`registerEVSE` Adds the location of the EVSE with price and link to the rest of the information.

`getEVSE` Returns all available info of the EVSE.

`UnableEVSE/disableEVSE` Makes the EVSE visible/invisble on the map.

`getAllEVSE` Returns the positions and ID (for the map).

Module *EVSE Pilot (Main Smart Contract)*

`Start Request` Coins are transferred, not passed in parameter ! This generated an event and stores the new TXId = TX[UserId][EVSEId].

`InterruptRequest` Request comes from the userId only. This generated an event.

`ChargeInterrupted` Transaction comes from the owner only. This generated an event and updates the status of the TXId. It sends coins to the user and to the owner.

`ChargeFinished` Transaction comes from the owner only. This generated an event and updates the status of the TXId.

`GetTxById` Returns the status of the transaction.

`GetTxBYUserId` Returns the transactions of the user and the status of these transactions. Uses a cursor to had them one by one.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

`GetTxByEVSEId` Returns the transactions of the EVSE and the status of these transactions. Uses a cursor to had them one by one.

Module *DAPP DEFev Creator*

`Login` Uses Gimly Login process to allow the user to see other pages.

`Dashboard` Displays the EVSEs of the user and his past history of transactions(charges).

`View EVSE` Detail EVSE from module DAPP DEFev.

`Create EVSE` Creates the EVSEBody of the EVSE: push the JSON to content storage and create EVSEId in EVSEAddress Book.

`my Account` Displays the address currently connected and the settings the user defined

Module *Community EVSE Manager*

`Main` When the EVSE replies OK, then EVSEMgr sends ChargeStarted to the Pilot. When the EVSE replies OK, then EVSEMgr sends ChargeInterrupted to the Pilot. After duration has expired EVSEMgr sends ChargeFinished to the Pilot

Module *ScoringBoard Smart Contract*

`AddComment score` Creates a score for the EVSEId, associated with the userId. The aggregated score is updated (total score and nb of scores). A text can be added.

`Read Aggregate Score` The aggregated score = average score and number of scores.

Module *Reputable BackEnd*

`Aggregate score` The aggregated score is computed and returned to the blockchain-based (average score and nb of scores). The unit score is also stored offchain for future features.

2.2.8 OTCnLNG

Description The OTCnLNG solution offers new capabilities to tackle issues actors are dealing with by generating transparent, traceable, accountable, secure data management for LNG buyers and sellers, responsible sourcing, and green LNG products. It includes REST API based webservice, ontology-based data structures, smart contracts,

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

and relies on the following external services: OriginTrail DKG for handling knowledge assets; and an EVM-compatible blockchain for deploying the OTCnLNG smart contracts.

Application Programming Interfaces They use the dkg.js, the Javascript SDK for using OriginTrail DKG and provide the API Specification for REST Services.

Module *dkg.js*

`DKG(params)` Connect to the DKG API.

`dkg.asset.create(content)` Create an asset on DKG.

`dkg.asset.get(UAL)` Retrieve an asset from DKG.

`dkg.graph.query(query, queryType)` Query the DKG.

Module *organizations* This enables to manage organizations and their users.

GET `https://<base_uri>/organizations/` Get the list of organizations.

POST `https://<base_uri>/organizations/` Create a new organization.

POST `https://<base_uri>/organizations/{orgDID}/generate-token` Create a registration token.

GET `https://<base_uri>/organizations/{orgDID}/registration/{token}` Create and register a user as part of an organization

Module *users* This enables to log as user within a given organization, and to list existing users.

GET `https://<base_uri>/users/` List all users.

POST `https://<base_uri>/users/login` Log in and get an auth token.

Module *wallets* This enables to manage the Ethereum wallets needed to interact with the resource and offset tokens.

GET `https://<base_uri>/wallets/` List all the wallets public addresses.

POST `https://<base_uri>/wallets/` Create (generate) a new Ethereum wallet (key pair stored on the platform).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Module *knowledge-assets* This enables to create and get knowledge assets.

GET `https://<base_uri>/knowledge-assets/` List all knowledge assets.

POST `https://<base_uri>/knowledge-assets/` Create a new knowledge asset.

GET `https://<base_uri>/knowledge-assets/{UAL}` Get the data for a given knowledge asset.

Module *dkg* This is used to query DKG.

GET `https://<base_uri>/dkg/` Query DKG.

Module *offset-tokens* This enables create and get carbon credit/offset token/information.

GET `https://<base_uri>/offset-tokens/` List all offset tokens owned by the users organization.

GET `https://<base_uri>/offset-tokens/{offset-token-URI}` Get the metadata related to a given offset token.

POST `https://<base_uri>/offset-tokens/` Tokenize an amount of mitigated carbon.

Module *resource-tokens* This enables to handle resource tokens and to manage association with carbon credit tokens.

GET `https://<base_uri>/resource-tokens/` List all resource tokens owned by the users.

GET `https://<base_uri>/resource-tokens/{resource-token-URI}` Get the metadata related to a given resource token.

POST `https://<base_uri>/resource-tokens/` Tokenize the LNG volume of a given cargo.

POST `https://<base_uri>/resource-tokens/{resource-token-URI}/lock` Lock offsets to a resource token.

POST `https://<base_uri>/resource-tokens/{resource-token-URI}/unlock` Unlock offsets from a resource token (only the owner can do it).

POST `https://<base_uri>/resource-tokens/{resource-token-URI}/transfer-to/{address}` Transfer some amount of resource tokens to a given address.

Module *cargos* This enables to handle specific knowledge assets: LNG cargos.

GET `https://<base_uri>/cargos/` List all cargo in the knowledge graph.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

GET `https://<base_uri>/cargos/{cargo-KA-DID}` Get the data for a particular cargo in the knowledge graph.

POST `https://<base_uri>/cargos/` Create a cargo KA from cargo data

Module *ghg-assessments* This provides means to handle emissions information (GHG statements) as knowledge assets.

GET `https://<base_uri>/ghg-assessments/` List all GHG assessments in the knowledge graph.

GET `https://<base_uri>/ghg-assessments/{ghg-assessment-KA-DID}` Get the data for a particular GHG assessment in the knowledge graph.

POST `https://<base_uri>/ghg-assessments/` Create a GHG assessment KA.

2.2.9 CREATE

Description UNITT Content Registry And Tokenized Exchange (CREATE), is a digital content marketplace that enables creators to distribute and monetize their creations in a trustworthy and transparent manner while ensuring privacy. The marketplace will be a solution that is intended to work with the ONTOCHAIN infrastructure and software and its future token(s), enabling content creators to exchange content for tokens from other users on a pay-per-view basis.

Application Programming Interfaces CREATE contains four REST API Controllers that provide the integrations to the Core Services, smart contracts, and ONTOCHAIN and other potential 3rd party services.

Module *SEARCH CONTROLLER* The Search Controller calls the smart contract search method and parses the results. The Search service can also use a cache to speed up queries.

GET `https://api.create.unitt.io/search/<keyword>` Search for published content with search parameters.

Module *CONTENT CONTROLLER* The Content Controller allows for: i) fetching content metadata from a given URL and constructs a preview result. ii) fetching full content with given id. iii) adding, updating and removing content with correct credentials (signatures).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

GET `https://api.create.unitt.io/preview/<url>` Preview online content meta-data.

GET `https://api.create.unitt.io/content/<id>` Resolve the full URL for the given content id. URL is used to fetch the actual content (text body, video stream etc.).

POST `https://api.create.unitt.io/content` Add new media content.

POST `https://api.create.unitt.io/content/sign` Sign the transaction with the user's key pair.

PUT `https://api.create.unitt.io/content` Update your media content.

PUT `https://api.create.unitt.io/content/sign` Sign the transaction with the user's key pair.

DELETE `https://api.create.unitt.io/content` Remove media content.

DELETE `https://api.create.unitt.io/content/sign` Sign the transaction with the user's key pair.

Module *PAYMENT CONTROLLER* The Payment Controller handles payments when purchasing content by consolidating transactions between ONTOCHAIN token and Pact smart contracts.

POST `https://api.create.unitt.io/payment` Pay for content.

GET `https://api.create.unitt.io/payment/sign` Sign the transaction with the user's key pair.

Module *USER CONTROLLER* The User Controller Registers subscriber devices and sends push notifications.

POST `https://api.create.unitt.io/notification` Subscribe to push notifications with current authorization header, FCM token and device data.

2.2.10 DAOstar

Description DAOstar is a set of open, semantic API standards for DAOs and DAO service providers, including EIP-4824 and DAOIP-3, built by a coalition of major DAOs, DAO frameworks, and DAO tooling providers. The goal of the DAOstar project is to develop standards that will ensure interoperability between both DAOs and DAO service providers.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Application Programming Interfaces Please refer to the API schema in DAOIP-3 Attestations for DAOs, which have been implemented in full as part of this project. Note that the standard has been updated based on their progress in the ONTOCHAIN project, including a signature field as well as a reputation-specific attestation type, to facilitate both REPUTABLE (which simply sums up reputation ratings) as well as potentially different contracts that perform some sort of reputation aggregation.

2.2.11 INGRESS

Description INGRESS provides access to cryptographically secured credit history for microlending in crypto and fiat currencies. The solutions addresses trust and security issues of digital economy using combination of biometric identification with asymmetric cryptography. This enables individuals possessing the private keys of re-issuing the credentials they own in unfortunate case of private key loss or compromise. INGRESS wallet connects the users to the marketplace of loans, provided by the lenders. Lenders compete among each other to win the users, which results in affordable credit for users with good credit histories.

Application Programming Interfaces INGRESS provide the APIs specification for Biometric SDK, Credit Bureau API and Biometric Bureau API.

Module *Biometric SDK* Biometric SDK module is used in both credit and biometric bureau REST services, as well as in the mobile application. This SDK generates the biometric sample from the biometric data. Even though its open sourced, Biometric SDK is a part of Iriscan background IP.

`BiometricSdk.configure()` Initial sdk configuration, should be called before anything else.

`BiometricSdk.getInstance()` Get sdk operations, thread safe.

`BiometricSdkConfigBuilder.withIris()` Enable iris recognition operations.

`BiometricSdkConfigBuilder.withFace()` Enable face recognition operations.

`BiometricSdkConfigBuilder.withFingerprint()` Enable fingerprint recognition operations.

`BiometricSdkConfigBuilder.build()` Build configuration object used for sdk configuration.

`BiometricSdkOperations.io()` Get IO operations (r/w biometric data, images, etc).

`BiometricSdkOperations.qualityControl()` Get image/biometric quality control operations.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

`BiometricSdkOperations.iris()` Get iris opertaions.

`BiometricSdkOperations.face()` Get face opertaions.

`BiometricSdkOperations.fingerprint()` Get fingerprint opertaions.

`InputOutputOperations.readRecord()` Read biometric record from bytes.

`InputOutputOperations.writeRecord()` Write biometric record as bytes.

`InputOutputOperations.readImage()` Read image from bytes.

`InputOutputOperations.writeImage()` Write image as bytes.

`QualityControlOperations.calulate()` Test image quality.

`IrisOperations.extractor()` Returns interface for extract operations.

`IrisOperations.encoder()` Returns interface for encode operations

`IrisOperations.matcher()` Returns interface for match operations.

`FaceOperations.extractor()` Returns interface for extract operations.

`FaceOperations.encoder()` Returns interface for encode operations.

`FaceOperations.matcher()` Returns interface for match operations.

`FpOperations.extractor()` Returns interface for extract operations.

`FpOperations.encoder()` Returns interface for encode operations.

`FpOperations.matcher()` Returns interface for match operations.

`IrisExtractor.extract()` Extract biometrics from image.

`IrisExtractor.extract()` Extract biometrics from native image

`IrisExtractor.extractRecord()` Extract biometrics from image.

`IrisEncoder.encode()` Create biometric template from image.

`IrisEncoder.encode()` Create biometric template from native image.

`IrisEncoder.extractAndEncode()` Extract and create biometric template from image.

`IrisEncoder.extractAndEncode ()` Extract and create biometric template from native image.

`IrisExtractor.encodeRecord()` Create biometric template from image.

`IrisMatcher.matches()` Match biometric templates.

`IrisMatcher.matches()` Match biometric template records.

`FaceExtractor.extract()` Extract biometrics from image.

`FaceExtractor.extract()` Extract biometrics from image.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

`FaceExtractor.extractRecord()` Extract biometrics from image.

`FaceEncoder.encode()` Create biometric template from image.

`FaceEncoder.encode()` Create biometric template from native image.

`FaceEncoder.extractAndEncode()` Extract and create biometric template from image.

`FaceEncoder.extractAndEncode()` Extract and create biometric template from native image.

`FaceEncoder.encodeRecord()` Create biometric template from image.

`FaceMatcher.matches()` Match biometric templates.

`FaceMatcher.matches()` Match biometric template records.

`FpExtractor.extract()` Extract biometrics from image.

`FpExtractor.extractRecord()` Extract biometrics from image.

`FpExtractor.encode()` Create biometric template from image.

`FpExtractor.encodeRecord()` Create biometric template from image.

`FpMatcher.matches()` Match biometric templates.

`FpMatcher.matches()` Match biometric template records.

Module *Credit Bureau* Component of the system to verify user credit score and manage loans. Endpoint: `https://credit-bureau.ontochain.iriscan.net/api/v1`.

GET* `/admin/admins/query` Get all admin users.

POST* `/admin/admins` Create new admin user.

PUT* `/admin/admins/{id}` Update admin user.

DELETE* `/admin/admins/{id}` Deactivate user.

POST `/admin/authenticate` Authenticate admin user.

POST `/admin/authenticate/restore-password` Request restore password.

POST `/admin/authenticate/restore-password/complete` Complete restoring password.

GET* `/admin/users/query` Get all users.

POST `/users/auth/register` Register new user in the system.

DELETE* `/admin/user/{id}` Deactivate user.

GET* `/admin/loans/query` Get all loans.

POST* `/admin/loans` Create new loan.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

PUT* `/admin/loans/{id}` Update loan information.
DELETE* `/admin/loans/{id}` Close loan.
GET* `/admin/loan-offers/query` Get all loan offers.
POST* `/admin/loan-offers` Create new loan offer.
PUT* `/admin/loan-offers/{id}` Update loan offer information.
DELETE* `/admin/loan-offers/{id}` Delete loan offer.
GET* `/admin/loan-applications/query` Get list of loan applications.
POST* `/admin/loan-applications` Create new loan application.
POST* `/admin/loan-applications/{id}/confirm` Confirm loan application.
POST* `/admin/loan-applications/{id}/deny` Deny loan application.
DELETE* `/admin/loan-applications/{id}` Delete loan application.
POST* `/admin/dumps` Initiate db dump.
GET* `/admin/dumps` Get all dumps.
GET* `/admin/dumps/{name}` Download zip dump.

Module *Biometric Bureau* Component of the system to creating/store and manage biometric credentials. Endpoint: `https://biometric-bureau.ontochain.iriscan.net/api/v1`.

GET* `/admin/admins/query` Get all admin users.
POST* `/admin/admins` Create new admin user.
PUT* `/admin/admins/{id}` Update admin user.
DELETE* `/admin/admins/{id}` Deactivate user.
POST* `/admin/authenticate` Authenticate admin user.
POST* `/admin/authenticate/restore-password` Request restore password.
POST* `/admin/authenticate/restore-password/complete` Complete restoring password.
GET* `/admin/users/query` Get all users.
POST* `/admin/dumps` Initiate db dump.
GET* `/admin/dumps` Get all dumps.
GET* `/admin/dumps/{name}` Download zip dump.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

POST* `/users/auth/register` Enroll new user.

POST* `/users/id/update-keys` Initiates keypair restore.

2.2.12 CAPS-CO

Description CAPS-CO has developed a practical application, a Carbon Accounting tool to calculate product carbon footprint (PCF) and implement a corporate carbon accounting (CCA) for manufacturers, resulting in transparent declared unit (DU) outputs based around the COP26 Pathfinder framework initiative (WBCSD, 2021). This tool delivers trusted, privacy-preserving, traceable, transparent, and legislation-compliant carbon accounting to European industry and incentivises emission-efficient operations.

Application Programming Interfaces The Core Application which implements API endpoint is built so that it can run both in cloud and on-premise, including a configuration where it is self hosted by the Business Entity. During the project, two versions of the CAPS-CO Carbon Accounting Application and API have been developed. Both v1 and v2 versions as currently they compliment each other are provided. This is being updated as we merge all the relevant endpoints into v2. Once this is done, v1 can be discarded.

Module *Companies - v1* This module allows the operation of Company entities.

POST `/v1/company/create` Creates a new company.

Module *Products - v1* This module allows the operation of Product entities.

POST `/v1/product/create` Creates a new product.

GET `/v1/product/getall` Retrieves all products.

Module *Activities - v1* This module allows to enter and retrieve business Activities. Activities data is used for CO2 emissions calculation.

POST `/v1/activity/create` Creates a new activity, initiates CO2 calculation and token issuance.

GET `/v1/activity/{productId}` Retrieves all activities by product ID

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Module *User (Wallet) Authentication - v1* This module allows to operate Users/Wallets entities.

POST `/v1/auth/checkWallet` Checks the wallet.
POST `/v1/auth/signup` Signs up the wallet.
POST `/v1/auth/signin` Signs in the wallet.
POST `/v1/auth/signout` Signs out the wallet.

Module *Activities - v2* In the latest version of CAPS-CO API, Activity service is extended and modified to implement the advanced logic of the solution. Support for audit is included.

POST `/v2/activity/create` Creates a new activity, logs inputs off-chain, calculates Inputs Hash and Components Hash, and initiates on-chain record.
GET `/v2/activity/{inputsHash}` Retrieves all activity details by a publicly known inputsHash.

2.2.13 Low-code app builder

Description The project within the ONTOCHAIN ecosystem consists of developing a low-code app builder to both ease the integration of different services of the ecosystem and make them more easily accessible to the end user by bundling them into an app. In other words, the solution can be thought of as the bridge between the different services of the ONTOCHAIN ecosystem and the end users. In more details, that bridge is an application builder to enable the different ONTOCHAIN services to be bundled and accessed via an app in a couple of clicks. To achieve this, they are partnering with a few projects of the ecosystem, namely Babelfish, Perun and PiSwap, in order to demonstrate how to integrate ones service into the low-code app builder.

Application Programming Interfaces The SDKs API is extensively described on the README page of its repository. Three specific APIs, each one associated with a specific service have been defined.

Module *Component registry subscription* An authenticated API allowing another ONTOCHAIN project to register its own component and associated data-sources. All components: `https://api.builderdev.startinblox.com/components/`. All components of a specific

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

user: `https://api.builderdev.startinblox.com/users/{{slug}}/components/`.

POST `/components/` Entrypoint for registering a new component in the registry.

GET `/components/` Returns the list of components, potentially filtered according to the query parameter.

GET `/components/{{slug}}/` Returns the metadata about the component identified by the slug parameter, including its URI.

PUT `/components/{{slug}}/` Updates the metadata about the component identified by the slug parameter.

DELETE `/components/{{slug}}/` Deletes the component from the registry

Module *Application builder engine* An authenticated API allowing a registered user on our platform to initialise, configure and deploy an app. All applications: `https://api.builderdev.startinblox.com/applications/`. All applications of a specific user: `https://api.builder-dev.startinblox.com/users/{{slug}}/applications/`

GET `/users/{{username}}/applications/` Returns the list of applications already initialised by username.

POST `/applications/` Adds a new app into the list of apps associated with the current user

GET `/applications/{{app-slug}}/` Returns the description of the applications created by a specific user, including their name, associated components, data-sources and visual template.

PUT `/applications/{{app-slug}}/` Returns the updated description of the application.

DELETE `/applications/{{app-slug}}/` Deletes the application from the registry.

Module *Deployment engine* An additional endpoint present on the application builder side and an asynchronous pull mechanism on the deployment engine side. Endpoint: `https://api.builder-dev.startinblox.com/ansible/inventory/` and `https://api.builder-dev.startinblox.com/ansible/slug/`.

GET `/ansible/inventory/` Returns a 200 - OK if everything went well or any kind of 4XX errors if there was a failure.

GET `ansible/slug/` Returns a 200 - OK if everything went well or any kind of 4XX errors if there was a failure.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

2.2.14 TRUSSIHEALTH

Description The TRUSSIHEALTH project proposes a decentralized and trustworthy health information exchange system. It leverages the concepts of Self-Sovereign Identity (SSI) and related technologies. In particular, TRUSSIHEALTH will develop a middleware that allows the conversation between health data in FL7 FHIR data format and the verifiable credential (VC) data format. VCs were specially designed to support SSIs by providing an open and lightweight data format used for storing and exchanging data. To add trust in the transformed health data, TRUSSIHEALTH will utilize the so-called eIDAS bridge, a tool that allows to apply qualified and advanced electronic signatures on VCs. This way, not only trust but also legal value is added to the transformed health data VC.

Application Programming Interfaces The TRUSSIHEALTH service offers two public API endpoints exposed by two modules, namely the import data module as well as the export data module, each of them provides one public endpoint. The other components cannot be called from outside the service but instead interact with different components such as the VIDchain API (backend) or the wallet backend.

Module *Import Data* This module is the main module of TRUSSIHEALTH responsible for the import data flow.

POST `https://labs.vidchain.net/health-data` Receive the health data object of a patient together with the DID related to it.

Module *Export Data* This module is involved in the sharing health data process and called by the verifier / relying party.

GET `https://labs.vidchain.net/health-data/<documentID>` Called by the data receiver after the user performed the verifiable presentation flow in order to receive the decrypted health data object.

2.3 ONTOCHAIN INTEROPERABILITY

The INTEROPERABILITY MODULES AND PROTOCOLS has multiple functions; first, it will act as the backbone for interconnecting the other modules; second, it will provide the ONTOCHAIN Gateway API, i.e. the main entry point for application developers; last, it will provide essential services and building blocks to all of the modules and to the applications, e.g. decentralised storage, identity management and data certification.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

This important layer has been mainly offered by the solutions developed during the first two years of the ONTOCHAIN project. As outlined in the D3.5 deliverable a and depicted in the Figure 1, it is clarified that the OC1 and OC2 projects contribute to the fulfillment of several modules of this layer. In particular, these projects establish the foundational framework for interconnecting the other modules, providing indispensable services and foundational elements to all of modules and applications. For more further details, we invite to refer to the D3.4 deliverable.

In this deliverable, we present the accomplishment of the INTEROPERABILITY MODULES AND PROTOCOLS layer's second objective, achieved by the BABELFISH project. Indeed, this project has successfully implemented the defined Gateway APIs, which facilitate a single entry point for upcoming developers and users. Section 3 describes all these Gateway APIs in detail.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

3 ONTOCHAIN GATEWAY API

The ONTOCHAIN Gateway API is the single-entry point for developers and users of ONTOCHAIN services. It is composed of three main modules:

1. **Service Discovery** lets programs search deployed ONTOCHAIN services with structured queries and access them directly.
2. **Accounts Management** provides functions related to user accounts and access rights.
3. **Data Storage** lets users and program upload data that can then be used by ONTOCHAIN services, and download certain data that has been produced by ONTOCHAIN services.

Each of these modules are already described in Section 3 of the D3.5 deliverable, along with the details of the API functions they provide.

These APIs, defined by the ONTOCHAIN consortium, were part of the Open Call 3 material and provided to the BABELFISH project selected in OC3. This section details the GATEWAY APIs implemented by BABELFISH.

Data stored via the Gateway can be stored using either on-chain storage:

- Bellecour: iExec Sidechain as the pilot infrastructure for ONTOCHAIN projects
- Convex: decentralised ledger provided by the non-profit Convex Foundation and/or off-chain storage:
- Semantic Container: transient data store for SMEs (see below)
- AWS S3: scalable, secure, and web-based cloud storage from Amazon

The API is documented in an HackMD document here:
https://hackmd.io/faNBTCUcSRyQsLOf_Jhdag?view.
The sources together with documentation is available on Github:
https://github.com/OwnYourData/dc-babelfish.
To interact with the REST APIs a Postman collection is available for import at the following location:
https://github.com/OwnYourData/dc-babelfish/tree/main/tutorial/postman

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

3.1 SERVICE DISCOVERY APIS

Service Discovery APIs provide a programmable way of discovering the services provided through the ONTOCHAIN ecosystem. The catalog provides search mechanisms and return pointers to the service implementations directly to the clients, in the form of API endpoints when available. The search features support advanced ontology-based queries so that consumer applications can search for services based on functionalities, implement their own selection algorithm and start consuming the services without requiring specific human interactions.

Module *Services catalog* This module will provide a programmable way of discovering the services provided through the ONTOCHAIN ecosystem. Access to the functions will be authenticated and regular users will only have access to read operations. Write operations will be accessible only to administrators, unless noted in the function's description.

GET `<base_uri>/list?page=X&items=X` *Retrieve Service Catalogue List (public).* Obtain a paged list of all available services; the filter can contain every field from resource description schema.
Arguments: `page` - selected page (default: `1`) and `items` - number of items per page (default: 20).
Return value: (array of json objects) services matching query.

GET `<base_uri>/service?query_field=query_value&field2=value2` *Query Service Catalogue List (public).* Obtain a list of services which name or description match the provided search terms.
Arguments: service catalogue query in the format `field, value`.
Return value: (array of json objects) services matching query.

GET `<base_uri>/service/SERVICE_ID` *Read Service Description (public).* Obtain details of a service description.
Arguments: `service_id` - numerical identifier of service.
Return value: JSON object with service description.

POST `<base_uri>/service` *Create Service Description.* Provide details of a service description and persist on the configured storage.
Arguments: `body` - JSON object with service description.
Return value: JSON object with name of the service and assigned service-id.

PUT `<base_uri>/service/SERVICE_ID` *Update Service Description.* Provide details of a service description and update on the configured storage.
Arguments: `service_id` - numerical identifier of service and `body` - JSON object with service description.
Return value: JSON object with name of the service and assigned service-id.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

DELETE `<base_uri>/service/SERVICE_ID` *Delete Service Description*. Mark service as deleted on the configured storage.
Arguments: `service_id` - numerical identifier of service.
Return value: JSON object with name of the service and assigned service-id.

3.2 ORGANIZATION AND USER ACCOUNTS APIS

Users in the ONTOCHAIN ecosystem can be either consumers or providers of services, or both. Users can be attached to an organization or be registered independently. The goal if this membership model is both to drive engagement, by providing ONTOCHAIN with specific features (e.g. a user portal) and to manage accounting, i.e. payment for services and crypto-wallets.

Module *Accounts management* This module provides interfaces for creating users and organizations, and basic interactions with wallets. All API calls must be authenticated (i.e. non public) and write operations are accessible only to administrators.

GET `<base_uri>/organization/ORGANIZATION_ID` *Read Organisation*. Obtain details of an organisation.
Arguments: `organization_id` - numerical identifier of organisation.
Return value: JSON object with organisation details.

GET `<base_uri>/organization/ORGANIZATION_ID/meta` *Read Organisation*. Obtain metadata of an organisation.
Arguments: `organization_id` - numerical identifier of organisation.
Return value: JSON object with organisation metadata.

GET `<base_uri>/organization/current` *Read Current Organisation* Obtain details of current organisation (based on provided Bearer Token).
Arguments: none.
Return value: JSON object with current organisation details.

POST `<base_uri>/organization/` *Create Organisation* Provide details of an organisation and persist on the configured storage.
Arguments: `body` - JSON object with organisation details.
Return value: JSON object with name of the organisation and assigned organization-id.

PUT `<base_uri>/organization/ORGANIZATION_ID` *Update Organisation* Provide details of an organisation and update on the configured storage.
Arguments: `organization_id` - numerical identifier of organisation and `body` - JSON object with organisation details.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Return value: JSON object with name of the organisation and assigned organization-id.

DELETE `<base_uri>/organization/ORGANIZATION_ID` *Delete Organisation* Mark organisation as deleted on the configured storage.
Arguments: `organization_id` - numerical identifier of organisation.
Return value: JSON object with name of the organisation and assigned organization-id.

GET `<base_uri>/organization/ORGANIZATION_ID/list` *Retrieve User List for Organisation* obtain a list of users for given organisation
Arguments: `organization_id` - numerical identifier of organisation.
Return value: (array of JSON objects) users of the organisation

GET `<base_uri>/user/USER_ID` *Read User*. Obtain details of a user.
Arguments: `user_id` - numerical identifier of user.
Return value: JSON object with user details.

GET `<base_uri>/user/USER_ID/meta` *Read User*. Obtain metadata of a user.
Arguments: `user_id` - numerical identifier of user.
Return value: JSON object with metadata of user.

GET `<base_uri>/user/current` *Read Current User* Obtain details of current user (based on provided Bearer Token).
Arguments: none.
Return value: JSON object with current user details.

POST `<base_uri>/user` *Create User* Provide details of a user and persist on the configured storage.
Arguments: `body` - JSON object with user details.
Return value: JSON object with name of the user and assigned user-id.

PUT `<base_uri>/user/USER_ID/` *Update User* Provide details of a user and update on the configured storage.
Arguments: `user_id` - numerical identifier of user and `body` - JSON object with user details.
Return value: JSON object with name of the user and assigned user-id.

DELETE `<base_uri>/user/USER_ID/` *Delete User* Mark user as deleted on the configured storage and remove all personally identifiable information.
Arguments: `user_id` - numerical identifier of user.
Return value: JSON object with name of the user and assigned user-id.

3.3 DATA STORAGE APIS

Data storage is a special kind of service in the way they provide a communication medium between other services. In order to facilitate these interactions, the Data

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

storage module provides high-level interfaces for storing and retrieving files and arbitrary blobs of data to and from providers, including external storage providers such as Amazon S3[3], Dropbox[4] and IPFS[5]. The underlying storage providers will be selected in the last year of the project, based on the requirements of the applications selected in OC3.

Module *Data storage* The module provides users and applications with high-level interfaces that are common to all of the supported backend storage services, both members of the ONTOCHAIN ecosystem, and external services. Storage providers can be searched in the Service catalog like any other service (see Section 3.1).

GET `<base_uri>/collection/list` *Read Organisation List*. Obtain the list of collection of objects.
Arguments: none.
Return value: (array of JSON objects) collection-id and name of collections.

GET `<base_uri>/collection/COLLECTION_ID` *Read Collection*. Obtain details of a collection.
Arguments: `collection_id` - numerical identifier of collection.
Return value: JSON object with collection details.

GET `<base_uri>/collection/COLLECTION_ID/meta` *Read Collection* Obtain metadata of a collection.
`collection_id` - numerical identifier of collection.
Return value: JSON object with collection details.

POST `<base_uri>/collection` *Create Collection* Provide details of a collection and persist on the configured storage.
Arguments: `body` - JSON object with collection details.
Return value: JSON object with name of the collection and assigned collection-id.

PUT `<base_uri>/collection/COLLECTION_ID` *Update Collection* Provide details of a collection and update on the configured storage.
Arguments: `collection_id` - numerical identifier of collection and `body` - JSON object with collection details.
Return value: JSON object with name of the collection and assigned collection-id.

DELETE `<base_uri>/collection/COLLECTION_ID` *Delete Collection* Mark collection as deleted on the configured storage.
Arguments: `collection_id` - numerical identifier of collection.
Return value: JSON object with name of the collection and assigned collection-id.

POST `<base_uri>/object/OBJECT_ID/USER_ID` *Check Object Access* Checks the access control for a particular user to access a particular object.

[3] https://aws.amazon.com/s3/
[4] https://www.dropbox.com/
[5] https://ipfs.io/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Arguments: `object_id` - numerical identifier of object and `user_id` - numerical identifier of user.
Return value: JSON object with name of object, assigned collection-id and object-id, and object access information.

GET `<base_uri>/object/OBJECT_ID` *Read Object*. (metadata) Obtain details related to and object.
Arguments: `object_id` - numerical identifier of object.
Return value: JSON object with object details (metadata).

GET `<base_uri>/object/OBJECT_ID/read` *Read User*. (object itself) Obtain the object.
Arguments: `object_id` - numerical identifier of object.
Return value: JSON object (object itself).

POST `<base_uri>/object` *Create Object* (metadata) Provide details of an object and persist on the configured storage.
Arguments: `body` - JSON object with object details.
Return value: JSON object with name of object and assigned object-id & collection-id.

PUT `<base_uri>/object/OBJECT_ID/write` *Write object* (object itself) Provide object and persist on the configured storage.
Arguments: `object_id` - numerical identifier of object and `body` - JSON object (object itself).
Return value: JSON object with name of object and assigned object-id & collection-id.

PUT `<base_uri>/object/OBJECT_ID` *Update Object* (metadata) Provide details of an object and update on the configured storage.
Arguments: `object_id` - numerical identifier of object and `body` - JSON object with-object metadata.
Return value: JSON object with name of object and assigned object-id & collection-id.

DELETE `<base_uri>/object/OBJECT_ID/` *Delete Object* Mark object as deleted on the configured storage.
Arguments: `object_id` - numerical identifier of object.
Return value: JSON object with name of object and assigned object-id & collection-id.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

4 PILOT NETWORK

The Bellecour sidechain has been adopted for the pilot use-cases of the ONTOCHAIN project. As already detailed in Section 4 of the D3.5 deliverable, this technology solves many well-known blockchain issues and offers new opportunies for using the blockchain technologies in various application domains. Moreover, the decision of relying on the EVM for the main ONTOCHAIN chain will ensure that upcoming updates to the Ethereum protocol and implementation will benefit past, current and future ONTOCHAIN services and applications. Indeed, since almost all ONTOCHAIN services sit at the application layer, they will remain compatible with Ethereum updates (to the exception of GraphChain which relies on a modified Hyperledger Besu client which may require a small update in order to support Ethereum 2.0).

The foundation and the developer's liabilities of the pilot network are detailed in Section 4 of the D3.5 deliverable. In this section, we provide the update of the pilot use-case, specifically the node deployment process and the current specification of these nodes.

4.1 NODE DEPLOYMENT PROCESS

As already mentioned in D3.5, the pilot network involves 8 validators and 8 fullnodes deployed in 4 countries: France hosted by iExec, Slovenia hosted by University of Ljubljana, Greece hosted by University of Athens and Italy hosted by IntelliSemantic.

The notable modification of these nodes is the client migration from OpenEthereum to Nethermind. Indeed, Openethereum client is becoming deprecated and Nethermind is the only production client that supports the AuRa consensus mechanism, used in the Bellecour sidechain. All node operators (fullnode and validator nodes) must migrate or be deployed their nodes to use Nethermind client. Unfortunately the migration cannot use the existing data of Openethereum client. Nethermind node needs to be synchronized from scratch.

The steps to do the migration are detailed at https://github.com/ONTOCHAIN/bellecour-node-deployer and are the following ones.

For full nodes:

1. In a new folder, different from the folder used for the old node, clone this repository:

```
$ git clone https://github.com/ONTOCHAIN/bellecour-node-deployer.git
$ cd bellecour-node-deployer/fullnode
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

2. Create a copy of the `.env.template` file and replace placeholders with your config.

```
$ cp .env.template .env
```

Use the same config values as the old node:

- `INSTANCE_NAME ==> NETSTATS_NAME`
- `CONTACT_DETAILS ==> NETSTATS_CONTACT_EMAIL`
- `WS_SECRET ==> NETSTATS_SECRET`

3. Run docker compose:

```
$ docker-compose up -d
```

4. Check the logs of the `bellecour-fullnode` container, you should see something like:

```
$ docker container logs -f bellecour-fullnode

...
2023-05-17 13:30:08.2468|Old Headers      32064 / 20069000 | queue  9024 | current
       0.00bps | total  6566.03bps
2023-05-17 13:30:08.2468|Downloaded 20072685 / 22156140 | current   888.69bps |
    total   924.79bps
2023-05-17 13:30:09.2458|Old Headers      43392 / 20069000 | queue  5376 | current
    11340.33bps | total  7380.70bps
2023-05-17 13:30:09.2458|Downloaded 20073320 / 22156140 | current   635.66bps |
    total   866.44bps
2023-05-17 13:30:10.0928|Discovered new block 22325588 13:30:10 (0x618474...278916)
    , tx count: 1 miner 0x02bbe17b1ee0e35d20d653fc39c8bb086af71d53, sent by [Peer|
    eth66|22325588|  51.136.117.95:30303], with AuRa step 336866042
2023-05-17 13:30:10.2453|Old Headers      51840 / 20069000 | queue  4032 | current
    8452.36bps | total  7536.96bps
2023-05-17 13:30:10.2453|Downloaded 20074084 / 22325588 | current   764.39bps |
    total   849.29bps
...
```

Which means that the node is syncing with the network. This operation takes time as the node will download all blocks of the chain from other peers.

5. The node should appear on Netstats interface https://netstats.bellecour.iex.ec.

6. Stop the old node and purge its data to free up some disk space. Go to the old fullnode folder and run:

```
$ docker-compose down --volumes
```

For validator nodes:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

1. The new Nethermind node should only start to sync once the old node is stopped. Go to the old validator folder and run:

```
$ docker-compose down
```

2. In a new folder, different from the folder used for the old node, clone this repository:

```
$ git clone https://github.com/ONTOCHAIN/bellecour-node-deployer.git
$ cd bellecour-node-deployer/validator
```

3. Create a copy of the `.env.template` file and replace placeholders with your config:

```
$ cp .env.template .env
```

   Use the same config values as the old node:

   - `MINING_KEY_ADDRESS ==> MINING_KEY_ADDRESS`
   - `INSTANCE_NAME ==> NETSTATS_NAME`
   - `CONTACT_DETAILS ==> NETSTATS_CONTACT_EMAIL`
   - `WS_SECRET ==> NETSTATS_SECRET`

4. Setup the mining key file. Copy the Mining key file in the same folder as the docker-compose and rename it to `validator-wallet-key.file`.

5. Setup the password file. Unlike Openethereum, Nethermind does not use env variables for wallet passwords. It uses files instead. So the content of the old variable `MINING_KEY_PASSWORD` should be saved in a file named `validator-wallet-password.file` next to `docker-compose.yml`.

6. Run docker compose:

```
$ docker-compose up -d
```

7. Check the logs of the container `bellecour-validator`, you should see something like:

```
$ docker container logs -f bellecour-validator

...
2023-05-17 13:30:08.2468|Old Headers      32064 / 20069000 | queue  9024 | current       0.00bps | total  6566.03bps
2023-05-17 13:30:08.2468|Downloaded 20072685 / 22156140 | current   888.69bps |    total   924.79bps
2023-05-17 13:30:09.2458|Old Headers      43392 / 20069000 | queue  5376 | current    11340.33bps | total  7380.70bps
2023-05-17 13:30:09.2458|Downloaded 20073320 / 22156140 | current   635.66bps |    total   866.44bps
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

```
2023-05-17 13:30:10.0928|Discovered new block 22325588 13:30:10 (0x618474...278916)
    , tx count: 1 miner 0x02bbe17b1ee0e35d20d653fc39c8bb086af71d53, sent by [Peer|
    eth66|22325588|  51.136.117.95:30303], with AuRa step 336866042
2023-05-17 13:30:10.2453|Old Headers      51840 / 20069000 | queue  4032 | current
    8452.36bps | total  7536.96bps
2023-05-17 13:30:10.2453|Downloaded 20074084 / 22325588 | current   764.39bps |
    total   849.29bps
...
```

Which means that the node is syncing with the network. This operation takes time as the node will download all blocks of the chain from other peers.

8. The node should appear on Netstats interface https://netstats.bellecour.iex.ec.
9. Remove the old validator's data to free up disk space. Go to the old validator folder and run:

```
$ docker-compose down --volumes
```

4.2 STATUS OF THE ONTOCHAIN NETWORK

The details specification of validator and full nodes have been updated in Table 1 and Table 2.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Validator Node Info	University of Ljubljana	IntelliSemantic	Athens University of Economics and Business	iExec
Number of deployed node	1	1	1	5
Hosted over Physical/Virtual platform	Cloud: The Academic and Research Network of Slovenia (ARNES)	Cloud: Linode provider	On-Premise	Azure Virtual Machine
Month of deployment	April 2022	May 2022	April 2022	September 2019
Up-Time for Validator	100%	100%	100%	100%
Hardware description	8GB RAM/100GB Storage/4 CPU cores	8GB RAM/160 GB Storage/4 CPU cores	16GB RAM/500GB Storage (SSD)/ 4 CPU cores	8GB RAM/16GB Storage (SSD)/ 2 CPU cores
Software dependencies	Docker and Docker-Compose, Veracrypt, Keys generation Dapp, PoA Smart Contracts	Docker and Docker-Compose, Veracrypt, Keys generation Dapp, PoA Smart Contracts	Docker and Docker-Compose, Veracrypt, Keys generation Dapp, PoA Smart Contracts	Docker and Docker-Compose, Veracrypt, Keys generation Dapp, PoA Smart Contracts
Number of Blocks Validated (mined)	916,207	890,807	714,357	4,776,740
Number of Transaction executed	185,092	193,062	113,559	1,015,255

TABLE 1: VALIDATOR NODES RELATED INFORMATION.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Full Node Info	University of Ljubljana	IntelliSemantic	Athens University of Economics and Business	iExec
Number of deployed node	1	1	1	5
Hosted over Physical/Virtual platform	Cloud: The Academic and Research Network of Slovenia (ARNES)	Cloud: Linode provider	On-Premise	Azure Virtual Machine
Month of deployment	February 2022	February 2022	April 2022	September 2019
Up-Time for Full node	100%	100%	100%	100%
Hardware description	16GB RAM/ 80GB Storage/ 4 CPU cores	8 GB RAM/160 GB Storage/4 CPU cores	16GB RAM/1TB Storage (SSD)/4 CPU cores	8GB RAM/16 Storage (SSD)/8 CPU cores
Software dependencies	Docker and Docker-Compose, PoA, NetStat	Docker and Docker-Compose, PoA, NetStat	Docker and Docker-Compose, PoA, NetStat	Docker and Docker-Compose, PoA, NetStat

TABLE 2: FULL NODES RELATED INFORMATION.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

5 CONCLUSION

This deliverable brought together the outcomes created by the thirty four projects, developed between March 2021 to August 2023. In particular, it provided the additional results offered by the OC3 projects to the ONTOCHAIN framework. The OC3 had two main objectives that have been successfully reached: completing the missing blocks of the ONTOCHAIN infrastructure and developing real-life use cases by including the ONTOCHAIN ecosystem. Thus, the final architecture of the ONTOCHAIN project is also presented and highlights the modules completed by all the ONTOCHAIN projects.

As described in the D3.5 deliverable, the OC1 and OC2 projects contribute to the implementation of several modules of ONTOLOGIES, DISTRIBUTED LEDGERS and INTEROPERABILITY MODULES AND PROTOCOLS layers. In addition, the OC3 projects enable to append new functionalities of the ONTOCHAIN framework, help for developing different components within the ONTOCHAIN framework and develop new real-life use case applications. In particular, the Gateway APIs module has been implemented by the BABELFISH project and the Convex Global DLT add a new distributed technology to the DISTRIBUTED LEDGERS layer. The other OC3 projects has successfully completed all the 9 modules of the APPLICATIONS layer.

Concerning the pilot network, there was no major modification made during this last year of the ONTOCHAIN project. In term of the deployed nodes, a seamless transition from Openethereum client to Nethermind has been carry out by the consortium. In this document, we recalled the deployment process of a validator and full node with the Nethermind client in this document. Moreover, this transition has not affected the performance of these node, since they continue to be available at 100% while effectively validating a considerable volume of transactions and blocks.

In conclusion, this deliverable described the specifications of the Gateway APIs and can be served as a reference of the final framework. Thus, this document could be provided to the future developers and users of the ONTOCHAIN ecosystem, willing use the ONTOCHAIN functionalities and services.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338