I-DELTA

Interoperable Distributed Ledger Technology

Deliverable 4.4

Platform implementation test results and technical documentation

Deliverable type:	**Document**
Deliverable reference number:	**ITEA| D4.4**
Related Work Package:	**WP4**
Due date:	**31.03.2023**
Actual submission date:	**24.03.2023**
Responsible organisation:	**KoçSistem**
Editor:	**Aylin Yorulmaz**
Dissemination level:	**Public**

Abstract:	The goal of this document is to explain the steps we take to make sure the product is completely functioning and ready for interoperability tests and distribution.
Keywords:	Blockchain technology, Interoperability, Atomic swaps, Blockchain integration, Distributed Ledger Technologies (DLT), Internet of Value, AI, ERP, IoT, P2P Energy Trading, Hyper Ledger platform, Local Grid, DApp, Tamper-proof, Green energy, Cloud Computing Platform, Edge Computing Platform, Implementation layers, Data Collection Service, Fabric Client App, Matching Engine, MQTT Broker, SemsPortal, REST API, Dart programming language, Flutter SDK, Mobile App, DLT Layer, Hyperledger Fabric, Smart Contract, PoC, Loyalty, Hyperledger Indy, Hyperledger Aries, Verifiable digital credentials

Table of Content

1. Executive Summary

In the context of blockchain technology, interoperability refers to the exchange of data between blockchain networks. Such data exchange may either include the replication of data from one blockchain to another, or the execution of functions conditional to information that was revealed on another blockchain. For example, transferring tokens between blockchain networks requires a locking operation on the source blockchain before minting respective representations on the target blockchain. Such proofs typically involve the replication of data to verify the locking event. On the other hand, exchanging tokens stored on distinct blockchains between two parties, so-called atomic swaps, includes revealing information to the counterpart who utilizes the learned fact to retrieve the exchanged asset.

While blockchain integration includes the orchestration and integration of blockchain clients into existing IT infrastructures, interoperability focuses on the communication between networks to exchange tokens, execute functions, or retrieve information.

2. Project Description

Distributed Ledger Technologies (DLT) undoubtedly are a cutting-edge new breed of technologies with the potential to completely transform the way our society works. DLT will foster switching from the “Internet of information" era to the “Internet of Value" era, whereby decentralized and immutable contracts define business interactions and secure exchanges of information. I-DELTA aims to create an interoperable DLT-based platform enhanced by AI, integrating with existing IT systems such as ERP and IoT applications.

3. Implementation

3.1. Components

3.1.1. P2P Energy Trading (Türkiye - KoçSistem)

This component is responsible for the development of first a simple prototype of the architecture based on a hyper ledger platform. A server has been configured as a full blockchain node at the cloud layer and another server was configured as a blockchain node but without a miner function, acting as a Local Grid LG-aggregator. A DApp (distributed application) software has been developed for mobile phones, allowing users to join the blockchain. Through the DApp, users can check the balance of their “energy wallet” after selling its power to the grid (making available power through a demand reduction action) or buying power from the grid. Users can also trace their usage of the energy by checking all the exchanged transactions with the Smart Grid. The figure below illustrates the prospective electricity usage control procedure for prosumers based on the dynamic electricity price and smart contract in the prototype. Orange lines denote the transactions within the blockchain overlay, while blue lines denote the data exchanges between the entities.

The blocks (ledgers) store, in a tamper-proof manner, the energy production and consumption-related data collected from intelligent meters. The self-enforcing smart contracts define programmatically the expected energy flexibility at the level of each producer and consumer, and thus, establish the rules for balancing the energy demand with the energy production at the grid level, keeping the grid system stable. In this way, blockchain provides a trustworthy and transparent means for encouraging green energy usage and reducing energy waste in a flexible and controllable way. For example, since the blockchain can trace all the transactions in the network and therefore knows the origin of every unit of electricity, a consumer can choose what type of energy he wants to use, e.g., from which generating plants or household he wants to purchase his electricity.

By introducing the blockchain overlay, the privacy-sensitive data, such as the value of meters, the type of each user's electrical equipment, and its geographical location, is encrypted and/or replaced by a public key, and will be stored, processed, and exchanged through the blockchain overlay. In addition, the transactions of each user, i.e., the amount and type of electricity power a user consumed or produced can be well traced. Note that not all the nodes in the blockchain overlay have the full function of a blockchain. Servers at the CCP – Cloud Computing Platform can maintain a full copy of the blocks and work as miners. Since the blockchain nodes can see all the transactions happening in the blockchain, only the trustworthy nodes can work as this type of node. For example, some of the routers with high computing power in the core network, or servers from the grid operator can be authorized to act as this type of node. The LG-aggregator at the ECP – Edge Computing Platform is a light blockchain node, it is trustworthy and can only store a copy of the blocks in the local areas. In practice, the definition and size of the area depend on the deployment of the architecture. The users' mobile devices are super-light blockchain nodes in that they can only exchange transactions with the grid, without storing blocks or being miners.

Implementation layers are presented below:

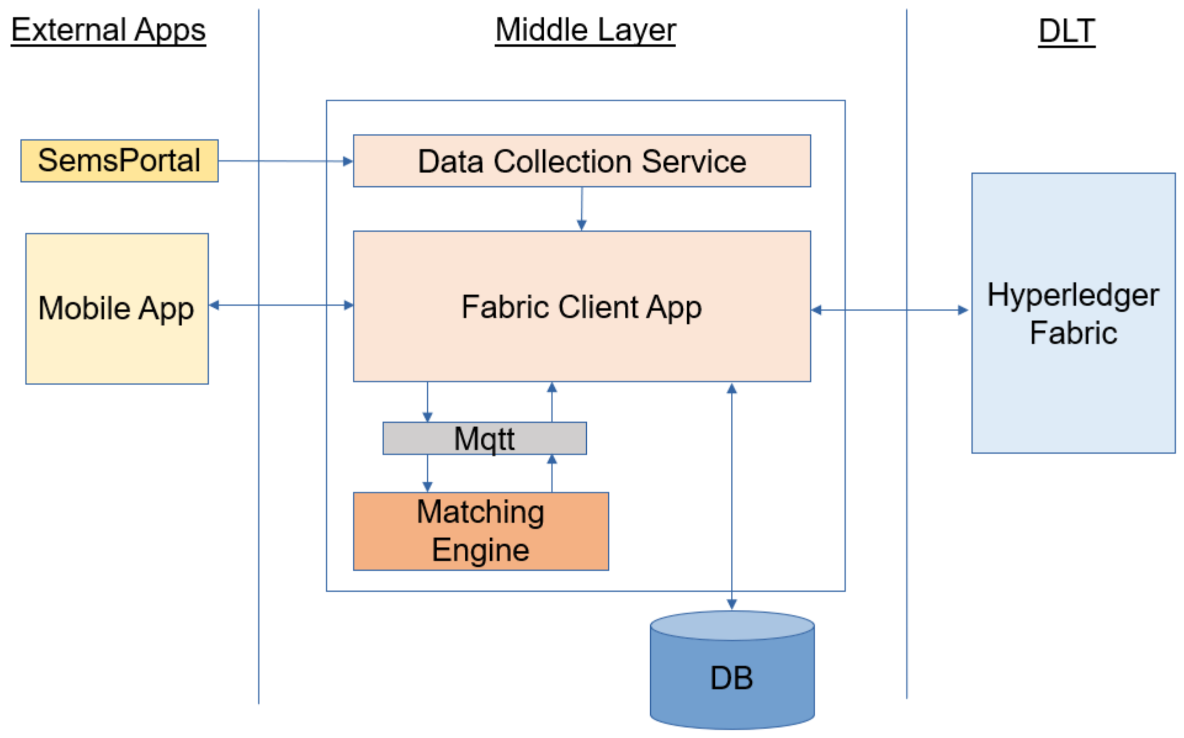


Figure 1. Implementation Layers Diagram

The figure above shows the different layers and components of the Energy Grid P2P trading PoC. Middle Layer components: Data Collection Service, Fabric Client App, and Matching Engine application are containerized using Docker. MQTT Broker and the database used by the system can also be deployed as Docker containers. The entire middle layer is designed to be deployed on Kubernetes on a multi-container pod.

3.1.1.1. External Application Integration

SemsPortal

SemsPortal application is a web server that is connected to the Solar Panels and Batteries that are used in this PoC. SemsPortal collects data in real time from these devices and serves them through REST API endpoints. Data Collection Service in Middle Layer collects this data periodically and updates the user and device information on the HyperLedger Fabric DLT and the database.

Mobile App

Mobile App developed using Dart programming language And Flutter SDK. End users will use this mobile app to perform peer-to-peer trading operations. The app provides multiple screens that are needed to perform the P2P use case.

- Devices Screen – Shows the devices that the user controls and their basic details.
- Orders Screen – Shows the list of buy or sell orders that the user has made. Along with the status of these orders such as completed, active, and expired.
- View Order Screen – Shows all the details related to the order and allows editing order details if possible
- Create Order Screen – Provides a form-style screen to create a buy or a sell order. Defining the amount of energy for trade, monetary details, and an interval for which the order will remain active.
- Marketplace Details Screen – Shows a price-over-time graph of energy trades that were completed within the PoC. Shows a summary table of all the trades.

3.1.1.2. Middle Layer

Data Collection Service

This service is developed using Node.Js. Using REST API, this service pulls data from SemsPortal for all the devices that are part of the system. The data is pulled in regular intervals and written to the DLT and the database. For this PoC, this interval is 1 hour. Data pulled from the service includes energy generated for solar panels or energy stored for batteries, all IoT data, and device information. Energy-related data for the related entities are updated on the Smart Contract. Detailed device information is stored on the database for Mobile App's usage.

Fabric Client App

This is a Node.Js server that will connect all the main components of the P2P energy trading system. The mobile app will send all the user operations directly to this app using REST API. Fabric Client app sends the information related to buy / sell orders is sent to a matching engine through an MQTT broker. It also listens to the MQTT broker for any trade matches that the matching engine finds. Fabric Client app handles all the required read/write operations to the database. This app implements the HyperLedger Fabric SDK to connect and call the Smart Contract functions.

MQTT Broker

An Eclipse Mosquito message queue is used to transfer data between the Fabric Client app and the matching engine.

Matching Engine

This application is used to find a match between buy and sell orders that were created by the end users. It is developed using Java programming language. As an order book and matching engine, the open-source “exchange-core” library is used.

Database

A PostgreSQL relational database that is used mainly to store off-chain information needed by the mobile application.

3.1.1.3. DLT Layer

HyperLedger Fabric version 1.4 is used as DLT. Smart Contract developed using Node.js is deployed on the Network. For this PoC, HyperLedger Fabric Network is set up using a single organization, single peer and 1 node Ordering Service and single Certificate Authority.

3.1.1.4. Loyalty

3.1.2. I-Benefit (Türkiye - Dakik Yazılım)

This project was started to expand the usage areas of the Loyalty project. It took its current form when it was desired to further expand these usage areas. A server configured for Hyperledger Aries, Hyperledger Indy and other web projects.

Hyperledger Indy provides tools, libraries, and reusable components for providing digital identities rooted on blockchains or other distributed ledgers so that they are interoperable across administrative domains, applications, and any other silo. Indy is interoperable with other blockchains or can be used standalone powering the decentralization of identity.
Hyperledger Aries provides a shared, reusable, interoperable tool kit designed for initiatives and solutions focused on creating, transmitting and storing verifiable digital credentials. It is infrastructure for blockchain-rooted, peer-to-peer interactions.

A web project, I-Benefit, has been developed for user’s verifiable credential management. Through this project, users will be able to see the verifiable credentials stored in Hyperledger Aries. Also, users can accept or delete the verifiable credential and see details of the verifiable credentials and download the verifiable credential as PDF to proof credentials offline.

Another web project, HepsiOrada, has been developed for shopping. Users can buy tokens on the Loyalty Project. With the integration, users can store their tokens as verifiable credentials in I-Benefit. Users can spend their tokens that are stored as verifiable credentials, in HepsiOrada. In the payment state, instead of current payment methods, HepsiOrada will want to prove your verifiable credentials that contain token information. If the credential is valid and it has more tokens than the price of the product, payment is complete. After the payment is done, the credential will be revoked. Users can not use revoked credentials again.

The last web application is Brutflix. Brutflix is a video streaming platform. Users can log in to Brutflix with a credential that is issued from the Loyalty application. Verifiable credentials can replace username and password in our PoC. For the PG rated content, Brutflix uses Zero Knowledge Proof. With Zero Knowledge Proof, users can prove that they are suitable for PG rated content without revealing any information. If the user is older than 18, then Brutflix allows the user to watch. The figure below shows interaction between Loyalty application and usage areas.

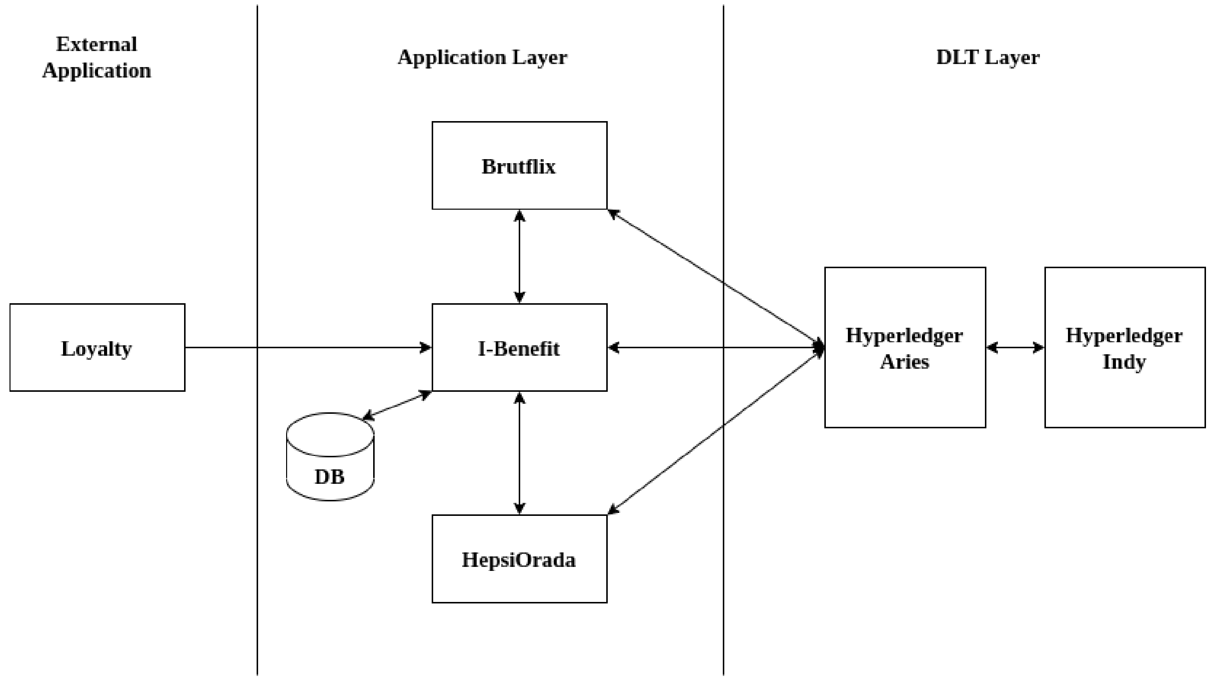


Figure 1. Implementation Layers Diagram

The figure above shows interaction between layers. Hyperledger Indy only interacts with Hyperledger Aries. But Hyperledger Aries interacts with Brutflix, I-Benefit and HepsiOrada. These applications have agents in Hyperledger Aries. On the other hand Loyalty only interacts with I-Benefit. I-Benefit is the center of our PoC. Whenever HepsiOrada or Brutflix wants to interact with a user's wallet, they must interact with I-Benefit.

3.1.2.1 DLT Layer

Hyperledger Aries
To interact with Hyperledger Indy, 5 Hyperledger Aries agents named I-Delta, Government, Brutflix, HepsiOrada and Multi-Tenant were created. Every agent has a different purpose. For example, Multi-Tenant agent is created for users. Whenever a user logs in the I-Benefit for the first time, a wallet is created under the Multi-Tenant agent. Agents must have a connection between them to interact with. Every agent containerized using Docker.

Hyperledger Indy
Hyperledger Indy used as DLT. Indy network is created with Sovrin Verifiable Organizations Network and has 4 Validator Nodes. The role for these Validator Nodes is Steward. Indy network contains DIDs, Schema IDs, Credential Definitions IDs and Revocation information.

3.1.2.2 Application Layer

I-Benefit
This application is developed using React and JavaScript. Using REST API, I-Benefit communicates with Hyperledger Aries Agents whenever a user interacts with any agent. For example, while the user accepts a credential, the credential request is kept in the Multi-Tenant agent. After the user accepts the credential in the interface, the acceptance message is sent to the Multi-Tenant agent and performs the operation through the connection between the agent that assigned the credential and the Multi-Tenant agent. I-Benefit has access to all endpoints in the Multi-Tenant agent. But it has access to some endpoints in the rest agents. These endpoints are mostly connection endpoints. Because in Hyperledger Aries, establishing a connection is a two-sided operation.

HepsiOrada
This application is also developed using React and JavaScript. HepsiOrada has only access to HepsiOrada agent endpoints and database endpoints which store the products and transactions. Whenever HepsiOrada wants to interact with users' wallets, it must redirect to I-Benefit because HepsiOrada has no access to users' wallets. For example, when user buys a product, in the payment state, HepsiOrada creates a proof to request that user's credential information. Since HepsiOrada has no access to users’ wallets, it redirects to I-Benefit for user confirmation.

Brutflix
Brutflix has same idea with HepsiOrada. It developed using React and JavaScript too. It also has access to Brutflix agent and database endpoints. Users can login to Brutflix with their username and password or credential. When a user login, Brutflix creates a proof to reveal the expiration date from a specific schema and redirect to I-Benefit. After the user accepts the reveal, Brutflix verifies the proof and if it meets the requirement, it will be granted access to the application. For the Zero Knowledge Proof, Brutflix creates different proof to verify the user is

older than 18 and redirect to I-Benefit for user confirmation. But the Multi-Tenant agent will not reveal the date of birth.

Database

A MongoDB NoSQL database that is used for authentication, products, orders and movies for the web applications.

3.1.2.3 External Application Integration

Loyalty

I-Delta agent is created for integration. Loyalty application can create credentials with this agent. When a user buys a benefit from Loyalty application, it sends a request to I-Delta agent endpoints. Many of the credentials in I-Benefit, comes from the Loyalty app.

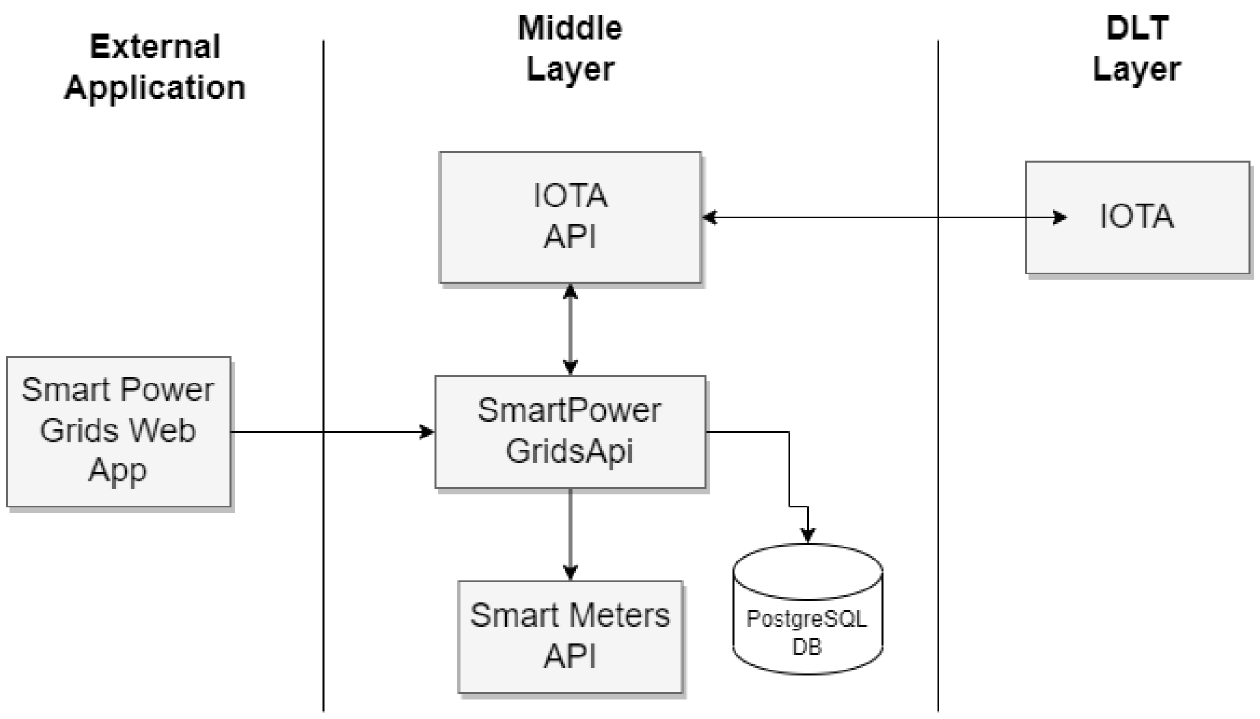
3.1.2. Smart Power Grids (Spain – Softtek, Sotec)

Smart Power Grids enables communities to create Smart grids without depending on a centralized entity. Each Smart grid has an administrator who manages the users. Each user has a Smart meter that records energy consumption. Energy consumption/production transactions are carried out through smart contracts, in which users agree on the price of energy purchase/sale. When the contract ends, each user receives tokens corresponding to their consumption/production.

Users have access to a web application where they can view their energy consumption/production, as well as their wallet and transactions made. Administrators can manage users, smart contracts, and the behavior of the Smart grid they administer.

To ensure the immutability and security of transactions, they are stored on the IOTA Tangle. The IOTA Tangle enables greater scalability than traditional blockchain solutions, allowing for more efficient and faster transactions. This is particularly important in the context of Smart Power Grids, where a large number of Smart meters generate vast amounts of data that need to be stored and processed.

By leveraging the benefits of the IOTA, Smart Power Grids provides a reliable and secure platform for communities to manage their energy consumption and production. It empowers users to make informed decisions about their energy usage and incentivizes them to reduce their carbon footprint, contributing to a more sustainable future. The figure below shows interaction between layers.



3.1.2.1. External Application Integration

Smart Power Grids Web App: This application is developed using Angular, providing a user-friendly interface and seamless experience. Both consumer and producer users can utilize this app to monitor their energy consumption and production behavior in real-time, enabling them to make informed decisions about their energy usage. Additionally, users can review their token-based transactions, ensuring transparency and building trust within the community.

Administrators have a comprehensive set of tools at their disposal to manage users who are part of the microgrid. They have the ability to create smart contracts, configuring parameters such as pricing, duration, and other conditions, allowing for the customization of agreements based on the specific needs of the community. Furthermore, administrators can view transactions made within the microgrid, track green energy consumption, and access historical pricing data, supporting data-driven decision-making and fostering the growth of sustainable energy practices.

3.1.2.2. Middle Layer

Database The PostgreSQL-based database is designed to store user authentication data, microgrid information, and smart contract details. It also stores transaction data. To ensure that transactions have not been tampered with, periodic audits are conducted using the Tangle. In the event of any alterations in the database, appropriate actions can be taken to address the issue.

Smart Power Grids API This REST API, developed in Python using Flask, interacts with the database and the IOTA API to record information in both the IOTA Tangle and the database. The Smart Power Grids API plays a crucial role in maintaining accurate records and streamlining processes, as it communicates with the Smart Meter API to retrieve smart meter readings for further analysis and decision-making.

IOTA API Developed in Python using Flask, this REST API enables various actions to be performed within the IOTA Tangle. It allows seamless interaction with the distributed ledger technology, enhancing the overall efficiency and security of the system. By providing access to the IOTA Tangle, the API supports the development and implementation of innovative solutions in the realm of energy management and beyond.

3.1.2.3. DLT Layer

3.2. Services

3.2.1. Services Implementation

3.2.2. Pilot Implementation

3.2.2.1. Pilot Applications

3.2.2.2. Alignment between Platform Services and Pilot Implementation

4. Quality Verification and Metrics

5. Testing

5.1. P2P Energy Trading Test Case Specification

The testing and evaluation of the P2P Energy Trading System is critical to ensuring that it meets the performance and usability requirements needed for real-world use cases. The testing methodologies and evaluation metrics used provide a comprehensive and objective assessment of the platform's capabilities. The results of the testing and evaluation provide valuable insights that can be used to optimize and improve the platform's performance, security, and scalability.

5.1.1. Description

Users specify their intention to participate in energy trading to the network by sending single-signature transaction representing bilateral contract. Users submit offers signed by their private keys individually. Users are notified about offer start and end time, the minimum price (MinP), and the maximum price (MaxP) allowed in the market via broadcast message, which is set for instance manually. The offers consist of energy quantity and price pairs for each trading interval I = 1, · · · , T P T I where TP (trading period) is equal to one full day consisting of 24 hrs.

5.1.2. Resources

The Validators validate transactions by checking several aspects including:

- User Authentication: User ∊ R (registered users).
- Validity of Price: MinP ≤ bid/offer price ≤ MaxP.
- Signature Verification: The transaction has been signed by the sender's private key.
- Account Balance: UAB of Consumer ≥ bid prices * bid energy amount.
- Production Capacity: PC of Producer ≥ offer energy amount.

5.1.3. Preconditions

There are two types of interactions taking place on the energy trading smart contracts: the external communication between users and the smart contracts; and the internal communication between smart contracts. Each of the smart contracts exposes a certain type of public interface for various functions such as Pool Market bid/offer submission, querying market clearing price, bilateral contract submission, balancing market offer submission, and payment settlement account balance query.

These public interfaces do not allow users to change information stored in smart contracts. The second type of communication is internal communication which happens via internal function calls between the smart contracts.

5.1.4. Evaluation Metrics

This section describes the performance evaluation metrics used in the experiments we carried out to evaluate the proposed system.

Write Transaction Latency (WTL): It involves a transaction that modifies the state of the blockchain such as bidding and market clearance. WTL is defined as the difference between transaction confirmation time (CT) and transaction submission time (ST). We consider that the timestamp of the block containing the transaction is equal to the initial CT of the transaction. For the consensus algorithms that provide immediate finality such as Hyperledger Fabric Raft, a transaction is considered committed the first time it appears in a block.

Read Transaction Latency: This is a transaction that only involves read operations such as checking account balance that does not modify the state of the blockchain. It is defined as the difference between submission and response time.

Write Transaction Throughput (WTT): This refers to the number of write transactions confirmed by the network per second. We calculate WTT as the number of successful transactions (NST) divided by the difference between the last confirmation time (LCT) and the first submission time (FST).

Read Transaction Throughput: This represents the throughput for query transactions. The same formula as WTT is used except that the CT is replaced by response time.

Scalability: This measures the change in latency and throughput as the number of transactions/nodes is increased.

Fail Rate: Transactions could get lost due to network congestion, node synchronization problem or large number of pending transactions causing overflow in the memory pool of the node. We define a failed transaction as a transaction for which confirmation is not received within 20 minutes which is higher than the current default waiting time in Ethereum (around 12 minutes). Since energy trading is a time-bound application, it does not seem reasonable to wait more than 20 minutes. The fail rate is the percentage of failed transactions.

Performance: We aim to monitor the peak performance at the initial network size of 20 nodes. The peak latency and throughput for the PoW algorithm.

Scalability: To evaluate the scalability of the system, we aim to monitor the change in latency, throughput, and fail rate by varying the number of transactions from 1000 to 10,000 and the number of nodes from 20 to 60. Plus, we aim to test the effect of increasing the number of transactions and number of nodes.

5.1.5. Flow of Events

We aim to benchmark the performance of our proposed unified energy trading model with the non-unified energy trading model with respect to the number of payment settlement transactions and transaction latency as we vary the number of traders. The non-unified energy trading model performs payment settlement for each market and for each user separately. This results in a redundant number of transactions which affects the performance of the network. On the other hand, the unified model handles payment settlement in a single point for the market. The number of transactions for the unified model is going to be low compared to the nonunified model.

5.1.6. Expected Results

The evaluation results are expected to indicate that Blockchain-based P2P systems perform well at a lower transaction rate of up to 4000 transactions per minute. However, the performance is expected to start to degrade as the number of transactions increases beyond that. Increasing the number of transactions and/or the number of nodes has little effect on read transactions as compared to write transactions. Moreover, different algorithms responded differently to the change in the number of transactions and the number of nodes.

5.2. I-Benefit Test Case Specification

I-Benefit has different test case scenarios. Testing is important for I-Benefit. For real-world use cases, it is necessary to meet the requirements. Since I-Benefit is an extensible application and to give an example, an external platform can be included in I-Benefit after creating the Hyperledger Aries agent and making the necessary configurations. Because of that we split our test cases like this;

- Ledger Test Case
- Login Test Case
- Connection Test Case
- Issue Credential Test Case
- Proof Test Case

5.2.1. Ledger Test Case

5.2.1.1 Description

Hyperledger Aries agents can interact with Hyperledger Indy's ledger. Hyperledger Indy Ledger is our core part in PoC. Hyperledger Aries agents will issue or proof credentials via Hyperledger Indy. For example, to issue a credential, the schema of the credentials must be written in

Hyperledger Indy already and the agent will read the schema from there. Agents can't issue a credential which is not written in the Hyperledger Indy.

5.2.1.2 Flow of Events

Someone with control over the agent sends a request to the Hyperledger Aries agent's endpoint. Hyperledger Aries receives this request then sends it to Hyperledger Indy to write to the ledger. Hyperledger Indy validates the request with 4 Validator Nodes. After that, schema will be added to the Ledger with Schema ID and Credential Definition ID. If the person who sends the request to Hyperledger Aries, defines the schema will supports revocation, Hyperledger Indy will prepare a rev_reg_id using schema id and credential definition id. This ID is the ledger equivalent of the accumulator information required for revocation. After these transactions, Hyperledger Indy will return Schema ID and Credential Definition ID to Hyperledger Aries.

5.2.1.3 Expected Results

Write transactions to Hyperledger Indy is the most time consuming part in I-Benefit. The fewer transactions means the faster in most blockchain-based systems. Hyperledger Indy is one of them. Writing a schema to the ledger takes 18 seconds. Writing a DID to the ledger takes 7 seconds. But read transactions take much less than that and have much less effect on the number of transactions.

5.2.2 Login Test Case

5.2.2.1 Description

In this test case, we tested the performance when users login to I-Benefit. There are two scenarios. One of them is when the users log in for the first time and the second one is when users log in before that. In the first scenario, when the user logs in for the first time, the DID of her/his wallet is written to Hyperledger Indy. In the second scenario, the application will get connection ids with agent and credentials.

5.2.2.2 Preconditions

The database must contain the user's information and the user must have a wallet created in the Multi-tenant agent

5.2.2.3 Flow of Events

User enters her/his username and password in the login screen. I-Benefit checks the database for the users' information and if the user is valid and password is correct, in the first scenario, I-Benefit sends a request to the Multi-Tenant agent to create a DID and register it to Ledger. In the second scenario, I-Benefit sends a request to the Multi-Tenant agent to get connection ids between agents and credentials that are stored in the wallet.

5.2.2.4 Expected Results

We had two scenarios for the Login Test Case. As a result, we divided our tests into two parts and evaluated the results in two different ways. In the first scenario, the user was logging in for the first time, so their DID was not found in the ledger. Since a transaction was made in the ledger, the execution time of this scenario was longer compared to the second one. The total time for performing the database checks in I-Benefit and adding a transaction to the Ledger in the first scenario is over 10000 ms. The user can perform their tasks on the interface after 3000 ms. I-Benefit will wait for the write transaction in Indy to finish in the background. In the second scenario, the total time is 3250 ms.

5.2.3 Connection Test Case

5.2.3.1 Description

The user needs to establish a connection between the Multi-Tenant agent whose wallet is in and other Aries agents in order to interact. These connections are two sided. If a connection cannot be established, agents cannot operate between them.

5.2.3.2 Preconditions

The user must be registered and logged in to the system. At the same time, the user must have a wallet in the Multi-Tenant agent.

5.2.3.3 Flow of events

The user opens the Connection page by clicking on the Add Connection button located at the top right corner of the homepage. They then select the agent they want to connect with from this page and click on the Confirm Connection button. I-Benefit sends a request to the selected agent's endpoint, allowing them to create an invite and connection ID. It then sends the created invite to the Multi-Tenant agent. After receiving the invite, Multi-Tenant creates a connection ID on its own end. Upon receiving the connection ID from Multi-Tenant, I-Benefit sends another request to Multi-Tenant to accept the invite. After this request, Multi-Tenant sends a message to the agent that the connection is intended to be established via Aries, indicating that the invite has been accepted through the connection ID. Finally, I-Benefit sends a request to the established agent to accept the invite that Multi-Tenant has accepted. A message is sent to verify that the connection is active after it has been established.

5.2.3.4 Expected Results

We did not take into account the user's interaction time with the interface in the results section. After the user clicks on the Confirm Connection button, the average time it takes to establish a connection is around 850ms. This time difference between the agents that the user wants to connect to is so negligible that it can be considered insignificant.

5.2.4 Issue Credential Test Case

5.2.4.1 Description

I-Benefit's purpose is to securely store issued credentials in the wallet and share them with other agents with the user's consent when necessary. For these operations to occur, the credential must be issued by another agent. In this test case, we focused on the issuance of the credential.

5.2.4.2 Preconditions

For the issuance of a credential to be possible, the user must be registered and logged into the system. Additionally, the Multi-Tenant agent must have a wallet and must have established a connection with the agent who will issue the credential. The agent issuing the credential must have written the schema for the credential to Indy.

5.2.4.3 Flow of events

Optionally, the user can request a credential from the issuer agent. In addition, after any action taken by the user, the issuer agent can prepare a credential and send it to the user's wallet via the connection established with Multi-Tenant. For example, if the user purchases a benefit from the Loyalty application, the Loyalty application, which has authorization on the I-Delta agent, creates a credential for the purchased benefit and sends it to the user. After the user accepts this credential, they start storing it in their wallet.

5.2.4.4 Expected Results
In this test case, we only considered the time it takes to send the credential and what happens after the user accepts it on the interface. We did not take into account the preparation of the information inside the credential or the time it takes for the user to accept the credential from the user interface. The time it takes to send the created credential to the Multi-Tenant agent via Hyperledger Aries is an average of 270 ms. The time it takes for the user to accept the credential and store it in their wallet is an average of 160 ms.

5.2.5 Proof Test Case

5.2.5.1 Description
The user can share the credential in their wallet with an organization or platform's agent that requests it. The organization or platform must specify the information they require beforehand. If the user approves the request, the information requested by the organization/platform will be shared from within the credential. The organization/platform can verify the information through their created agent.

5.2.5.2 Preconditions
The user must be registered and logged into the system. Additionally, they must have a wallet in the Multi-Tenant agent and establish a connection with the issuing agent. Hyperledger Aries has a proof feature that does not require a connection, but we did not use this feature in our project. The user must keep the credential requested for proof in their wallet.

5.2.5.3 Flow of events
To access the credential information, the agent requesting it prepares a proof and sends it through the connection between the Multi-Tenant agent. After the user approves this request, they select the credential they want to share and send it to the Multi-Tenant agent. The Multi-Tenant agent then retrieves the requested information from the selected credential and sends it to the requesting agent. The agent requesting the credential information verifies the information received from the Multi-Tenant agent.

5.2.5.4 Expected Results
In this Test Case, we focused on the time it takes for the agent requesting access to credential information to create a proof, the time it takes for the Multi-Tenant agent to perform its operations after the user selects the credential, and the time it takes to verify the information when sent between agents. The average time it takes to create a proof and send it to the Multi-

Tenant agent is 130ms. The time it takes for the user to select the credential is not taken into account. However, after the selection is made, the time it takes for the Multi-Tenant agent to process the request is 580ms on average. The time it takes for the agent to verify the information received is 480ms on average. As the number of requested information increases, the time it takes may also increase. However, this difference is insignificant and will not be recorded.

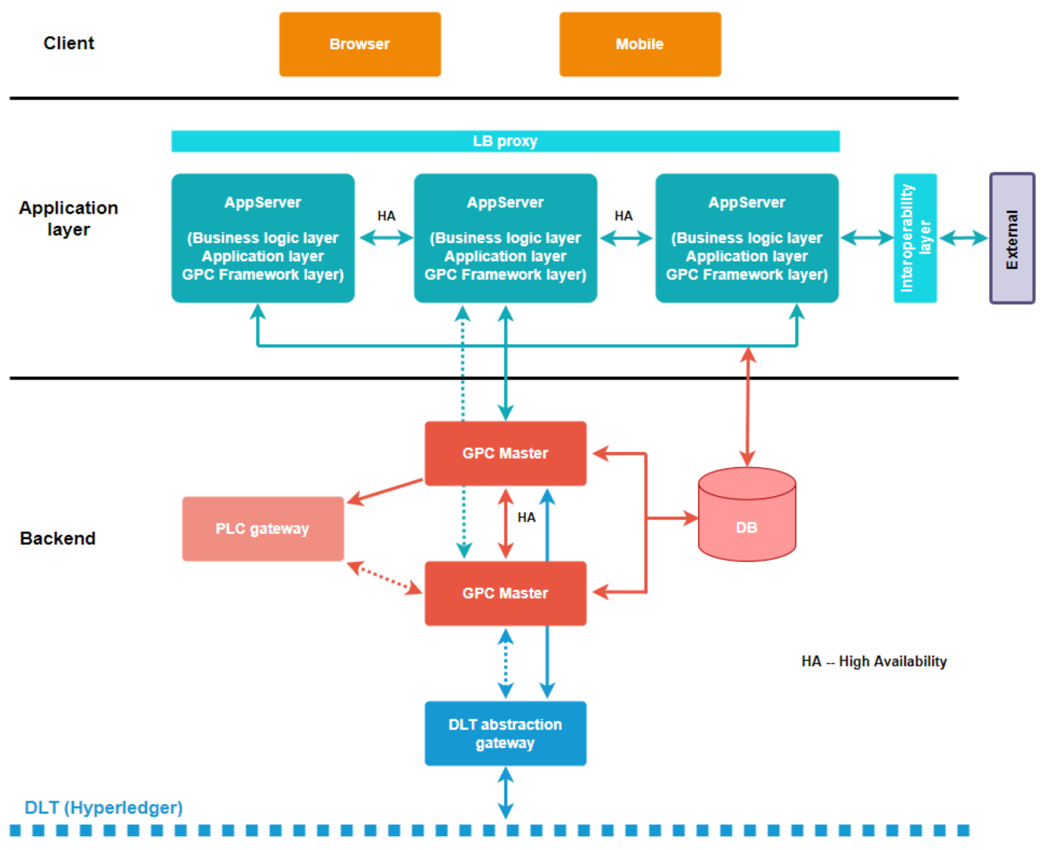
5.3. Legal agenda use case Test Case Specification

The voting application enables per-rollam voting at the general meeting in companies and organizations. The application provides a web interface for desktop computers, tablets and mobile phones. Most importantly, it offers:

- to call a per-rollam vote,
- to specify the individual items to be voted on, including the possibility to add accompanying electronic materials,
- to choose the method of evaluation of the vote,
- and to publish them in the DLT Hyperledger Fabric.

An automated part of convening a vote is the transactionally secure creation of voting tokens for each eligible voter, which occurs within the DLT via chain-code.

Diagram of layers and use case components:



For **the client layer**, both browser and mobile view is available. These two components are backed also by our voting device (connected via PLC gateway).

The application layer includes individual sub-applications, functionalities and support services operated within the platform or offered by external / service providers with which the platform is connected. The app server contains the specific implementation of the voting workflow.

At the backend the database is built on PostgreSQL. As for the voting transactions, they can be written into the internal Hyperledger Fabric using the GPC Master that serves as the main decision unit. This layer also contains a gateway for connecting several hardware modules (sensors, actuators, display devices, etc.), in this test case the voting device.

The workflow used for testing:

By signing into the application a ballot administrator creates a request for remote voting (per-rollam). For the subsequent step the user sets properties of the voting (definition of the voting). These properties include how big the majority of voters must be in order to have a valid result, voting period, description, additional conditions for valid vote etc. If the number of available participants is sufficient a proposal statement with generated electronic ballots are created. Ballots are then committed into our Hyperledger Fabric, making it available for eligible voters for a defined time period.

5.3.1. Testing of the legal agenda application for voting at general meetings

As the last project phase was focused on integration of individual components, we have conducted tests of communication between the application and the Hyperledger Fabric blockchain, as well as the secured connection with the hardware voting devices.

Testing of functional requirements provided by the law firm KUBEČKA & PROKOP involved checks of digital voting process, the user interface, communication protocols and again the integration with the hardware voting devices (now from the end-user perspective).

In relation to performance tests, we have put the application under a large number of access requests to ensure that it can handle a high number of users and transactions at one time. This includes testing the application's response time, throughput, and scalability (automated tests using the Selenium IDE).

Security tests covered encryption and authentication mechanisms, as well as the application's ability to prevent unauthorized access and tampering of the voting data.

5.3.2. Testing of the hardware module for identification of a person and execution of legal agenda

A new firmware was tested for the HW module to control the connected HMI components (display, rotary controller, fingerprint reader, voting buttons, signal LED) and the integrated HW cryptographic operations accelerator.

In addition, the device runs a multi-threaded application that allows integration of the module with the I-DELTA ecosystem. This is an agent that allows local execution of selected elements of the BPMN steps that define legal actions. The device was tested for internet connectivity, both wire-based (Ethernet) and wireless (WiFi).

To secure communication and simplify access to I-DELTA, a VPN was tested with unique pre-configuration for each device – the device receives a generated custom TLS key, signed by a certificate authority. This also makes it easy to deny access to a compromised device by simply revoking the TLS key. The device is capable of making DLT calls, but the actual network node does not run on it.

5.4. Smart Power Grids Test Case Specification

Smart Power Grids aims to manage energy trading through technologies like IOTA and smart contracts. In such a system, accuracy and reliability are essential to ensure that energy transactions are carried out effectively and without errors. Software testing helps detect potential flaws in the code and system functionality, significantly reducing the risk of errors in energy transactions and increasing confidence in the system.

- Transaction Recording
- Hash Verification

5.4.1. Transaction Recording

5.4.2. Description

In this test case, the goal is to ensure proper recording of energy consumption and production readings in the smart contract and IOTA Tangle. The results of each transaction's registration should be stored in a database.

5.4.3. Preconditions

Prior to this, the microgrid administrator must have created the user with the email and smart meter identifier. The administrator must have created the smart contract, and it must be active.

5.4.4. Flow of Events

The smart meter sends energy consumption and production readings through the system. The meter reading is correctly stored in the Smart Contract and the database. It is verified that the transaction has been registered in the IOTA Tangle and that a unique hash has been generated. The transaction information can be queried in the database.

5.4.5. Expected Results

The transaction is correctly registered, and the energy consumption and production information in the system is updated. This test case ensures that energy consumption and production information is correctly recorded at all system layers and securely and reliably stored in the Smart Contract, the database, and the IOTA Tangle. Additionally, it verifies that the information can be queried later in the database.

5.4.6. Hash Verification

5.4.7. Description

In the context of a system that uses blockchain and IOTA Tangle technology to record energy consumption and production transactions, it is essential to ensure that the information stored in the database is accurate and in line with the records in the IOTA Tangle.

5.4.8. Preconditions

The microgrid administrator must have previously created a user with the email and smart meter identifier.

The administrator must have created and activated the smart contract.

Smart meters must have sent readings to the system.

5.4.9. Flow of Events

For each smart contract, the database is queried to retrieve energy transactions data, including the amount of energy produced, energy consumed, the agreed-upon energy price, and the transaction hash. Using the hash of each record, the IOTA Tangle is queried, and the information retrieved from the database is compared with the data recorded in the Tangle.

5.4.10. Expected Results

After querying the hash of each transaction, the API developed to communicate with IOTA is expected to return the following JSON:

```
{"message": {
"Arguments": {
"-consume": "0x0300000000000000",
"-produce": "0x0100000000000000"
},
"Contract Hname": "8e1a1672",
"Request": "4cVztuaagro6Hwdx9g2Zyq7ZZ48uFnREz51xVhmhqz7isxT",
"Request found in block": "44",
"Timestamp": "2023-03-17T10:52:40Z"
},
"result": "Successful"
}
```

The JSON object above contains two main keys: "message" and "result".

The "message" key contains a nested object with several attributes, including "Arguments", "Contract Hname", "Request", "Request found in block", and "Timestamp".

Within the "Arguments" attribute, there are the consumption and production readings, where "-consume": "0x0300000000000000" indicates a consumption of 3 units, and "-produce": "0x0100000000000000" indicates a production of 1 unit.

The "Contract Hname" attribute displays the hash of the contract in question, which is "8e1a1672".

The "Request" attribute indicates the hash of the transaction being queried.

The "Request found in block" attribute indicates the block number where the transaction was found in the IOTA Tangle.

The "Timestamp" attribute indicates the time the transaction took place.

The "result" key contains a text string that indicates the outcome of the query, which should be "Successful" if the query was successful.

If the information for each transaction does not match the records in the database, it will be considered that the database has been altered. Therefore, it is crucial that the information stored in the database is accurate and corresponds to the records in the IOTA Tangle to ensure data integrity and trust in the system.