PROJECT DELIVERABLE REPORT

Grant Agreement Number: 101058732

Joint Industrial Data Exchange Platform

Type: Delivery Report

D3.4 DLT platform components - Final

Issuing partner	Technovative Solutions (TVS)
Participating partners	University of Cambridge (UCAM) Universita di Trento (UNITN) Arteevo Technologies (AVO)
Document name and revision	D3.4 DLT platform components - Final
Author(s)	Miah Raihan Mahmud Arman (TVS) Tanvir Islam (TVS) Rasel Ahmed (TVS)
Reviewer(s)	Farhadur Arifin (TVS) Dr Feroz Farazi (UCAM) Simone Bocca (UNITN)
Deliverable due date	31/05/2024
Actual submission date	30/05/2024

Project Coordinator	Vorarlberg University of Applied Sciences
Tel	+43 (0) 5572 792 7128
E-mail	florian.maurer@fhv.at
Project website address	www.jidep.eu

Dissemination Level		
PU	Public	✓
PP	Restricted to other programme participants (including the Commission services)	
CO	Confidential, only for members of the consortium (including the Commission services)	
SEN	Sensitive, limited under the conditions of the Grant Agreement	

Table of Contents

List of figures

Executive summary

Distributed ledger technology (DLT) is a term that encompasses various systems that enable the creation, management, and exchange of digital assets without the need for a central authority or intermediary [1]. Based on the decision concluded in the deliverable report D1.3b, the JIDEP platform uses the Ethereum-based layer-2 (L-2) Polygon [2] blockchain network to develop and deploy our DLT-based functionalities. This report provides an overview of the main components of our DLT platform, how they interact with each other and the DLT platform-based functionalities that we will use in our project. The report also explains some of the vital phases of our DLT platform components.

Some of the main aspects of our DLT platform are consensus mechanisms, smart contracts, scalability, and interoperability. The consensus mechanism is the process by which the nodes in a DLT network agree on the validity and order of transactions and blocks. Smart contracts are self-executing programs that run on the DLT network and can enforce rules and conditions [3]. Scalability is the ability of a DLT platform to handle increasing amounts of transactions and users without compromising its performance or security. Interoperability is the ability of a DLT platform to communicate and exchange data with other DLT platforms or external systems.

The report briefly overviews the main findings and reasons for choosing the suitable framework, platform, and libraries for the different phases of building and deploying our DLT platform-based service for JIDEP use cases.

The report sheds light on our data anchoring mechanism. In JIDEP, the digital assets that we are concerned about are the Material Passports, which will carry detailed information about the products of our different use cases. We store the data in a distributed manner and store a cryptographic hash of the data in the blockchain through our deployed smart contract. The hash of any material passport can be generated, and then we can check the existence of the hash through our smart contract. Since distributed ledgers are immutable, we can ensure the authenticity and integrity of the data using this mechanism [4].

The report depicts our journey to find the best tools to write and deploy smart contracts by running scripts and exposing our DLT platform-based features through a RESTful API, which will be reported in the deliverable D3.5. This report also briefly discusses our DLT platform-based features and how we plan to address the challenges we will face in the pursuit of scaling up later.

1. Introduction

The current linear production and consumption model is causing environmental and social problems. The "take-make-waste" linear model relies on finite natural resources, generates waste and emissions, and contributes to climate change, pollution, biodiversity loss, and inequality [5]. A sustainable and circular "reduce-reuse-recycle" model can help industries address these challenges, improve efficiency and competitiveness, and contribute to sustainable development [6]. Adopting a circular model can mitigate negative impacts while creating positive impacts for stakeholders and the planet.

The JIDEP platform enables the exchange of industrial data to influence the future industrial landscape. Its strategic goals include promoting improved data exchange via ontologies, creating a cooperative ecosystem throughout the value chain, creating tools for industry sustainability, bolstering the independence and adaptability of European industries, and verifying the platform's effectiveness through practical use cases and demonstrations. JIDEP is positioned to foster industry-wide standard data practices while promoting innovation, efficiency, and collaboration.

Throughout the data exchange cycles, JIDEP plays a vital role as a platform for data interchange by ensuring the data's security, interoperability, integrity, and authenticity. While the distributed data storage system guarantees the shared data's security, JIDEP uses ontology services to enhance data interoperability. Our DLT platform ensures the validity and integrity of the data. In addition, we are developing various features on top of our DLT platform, such as data consumption tracking, provenance, and governance. This report briefly outlines our approach to creating DLT platform-based functionalities for JIDEP.

2. Background

2.1 DLT

Distributed Ledger Technology (DLT) is a system for storing and updating data across multiple nodes or computers without relying on a central authority or intermediary [1]. DLT allows for secure, transparent, decentralised sharing, synchronising, and data verification. However, blockchain is a type of DLT. Not all DLTs are blockchains. Blockchain is a specific way of organising data into blocks linked together by cryptographic hashes, forming a chain of records that cannot be altered or tampered with. Blockchain uses a consensus mechanism, such as proof of work or proof of stake, to ensure that all nodes agree on the validity of the data. Throughout this document, the terms blockchain and DLT are used interchangeably. The components of a blockchain include chains of cryptographically secured blocks for storing and validating data [7].

Cryptography: This is the application of mathematical functions to secure and verify data on the blockchain. Cryptography ensures data authenticity, immutability, and confidentiality and enables digital signatures, encryption, decryption, and hashing.

Peer-to-peer network: This distributed model connects the nodes or computers on the Blockchain. The peer-to-peer (P2P) network allows data to be shared, synchronised, and verified in a decentralised and transparent manner. It also enables communication and collaboration among different participants on the blockchain.

Ledger: This database records and stores the transactions or data on the blockchain. A ledger is organised into blocks linked by cryptographic hashes, forming a chain of records that cannot be altered or tampered with. Ledger also contains validity rules that define the acceptable practices within the blockchain.

Smart contracts: Smart contracts are self-executing contracts that can enforce rules and conditions without human intervention. They are programmed using code and stored on the Blockchain. Smart contracts can automate and streamline processes, such as transactions, agreements, or applications.

Consensus Mechanism: This is the method or protocol that ensures that all the nodes on the network agree on the same version of the ledger. Consensus algorithms prevent conflicts, errors, or attacks on the blockchain. Different blockchain networks use different consensus mechanisms, each with advantages and disadvantages [8]. Two of the most used consensus mechanisms in blockchain technology are:

Proof of Work (PoW): This consensus mechanism requires miners to solve complex mathematical puzzles, which consume much computational power and energy. The first miner who solves the puzzle can add a new block to the blockchain, and other nodes easily verify the solution. PoW is used by blockchains such as Bitcoin and Ethereum. The main benefits of PoW are that it provides high security, trust, and immutability and enables innovation and collaboration [9].

Proof of Stake (PoS): This consensus mechanism requires validators to stake a certain amount of coins or tokens to participate in the network. The validators are randomly selected to propose and validate new blocks and receive rewards for their work. PoS is used by blockchains such as Polygon, Cardano, Polkadot, and Ethereum 2.0. The main benefits of PoS are that it provides speed, efficiency, scalability, and environmental friendliness, as well as enables customisation and governance. We have used the Polygon blockchain network, which uses this consensus mechanism [10].

2.2 Public vs private DLT

Public blockchain technology is a form of distributed ledger technology that allows multiple parties to share and verify data without relying on a central authority. Public blockchains are open, transparent, and immutable, meaning anyone can join, view, and audit the transactions and records stored on the network. Public blockchains can also support smart contracts and self-executing agreements that enforce rules and conditions without human intervention.

We have used the Polygon (Matic) public blockchain network, a layer-2 solution based on Ethereum's most widely used layer-1 solution. It provides some advantages over using a private blockchain network. Some of the benefits are:

- **No Hardware Setup Needed:** Public blockchains eliminate the need for any organisation to set up and maintain its own blockchain hardware, which can be costly and complex.
- **Availability of Developer Resources:** Public blockchains like Ethereum have vast developer resources and community support, making it more manageable to find expertise and solutions to potential problems.
- **Decentralisation:** Unlike private blockchains that are inherently centralised, public blockchains offer true decentralisation, distributing control across a vast network of users.
- **Public Access to Data:** Our use cases require information to be publicly accessible, which aligns well with the transparent nature of public blockchains.
- **Time Efficiency:** JIDEP is a time-bounded project. Developing and maintaining a private blockchain is resource-intensive for a project with time constraints. Using an existing public blockchain allows for better allocation of developer time to other project features.

Polygon is explicitly chosen as a layer-2 solution that builds on Ethereum's established layer-1 technology. It offers additional benefits such as scalability and lower transaction costs, as detailed in the project's deliverable report D1.1.

2.3 Data anchoring

2.3.1 Overview

The data in the blockchain is immutable, and the same data is replicated over all the running nodes of a blockchain network. Due to the network's bandwidth limitations, keeping all the data in the blockchain is not feasible and affects scalability and operational costs. Therefore, only a portion of the data is stored in the blockchain, which points to the actual data stored elsewhere, known as data anchoring. One can consider the data as a ship, whereas the part of data that is kept in the blockchain is the anchor to that data. We can use this anchor to implement the DLT-based functionalities that ensure the different aspects of the data.
Moreover, the data staying on the blockchain is publicly readable. Disclosing sensitive data would be detrimental to the concerned stakeholders. So, keeping only a particular portion or form of the data avoids unauthorised data access while still ensuring the desired features [11].

2.3.2 Distributed data storage

The blockchain only stores a reference or anchor to the data, not the actual data itself. The actual data is stored off-chain, which means it's stored outside the blockchain. In our case, our distributed data storage system is off-chain storage, ensuring data is stored securely. The data is stored across multiple Distributed Storage Peers and backed up in a cloud server for maximum data persistence. More detailed information about this system is in the deliverable D2.7 report.

2.3.3 DSU

The report, D2.7, provides a detailed discussion of the Data Sharing Units (DSUs) used within the JIDEP platform. These DSUs facilitate data exchanges related to material passports within the platform. Whenever a Material Passport data creation or update operation occurs, a cryptographic hash of the data is computed, as illustrated in Figure 1. This hash is then stored in the layer-2 public blockchain, which anchors the data.

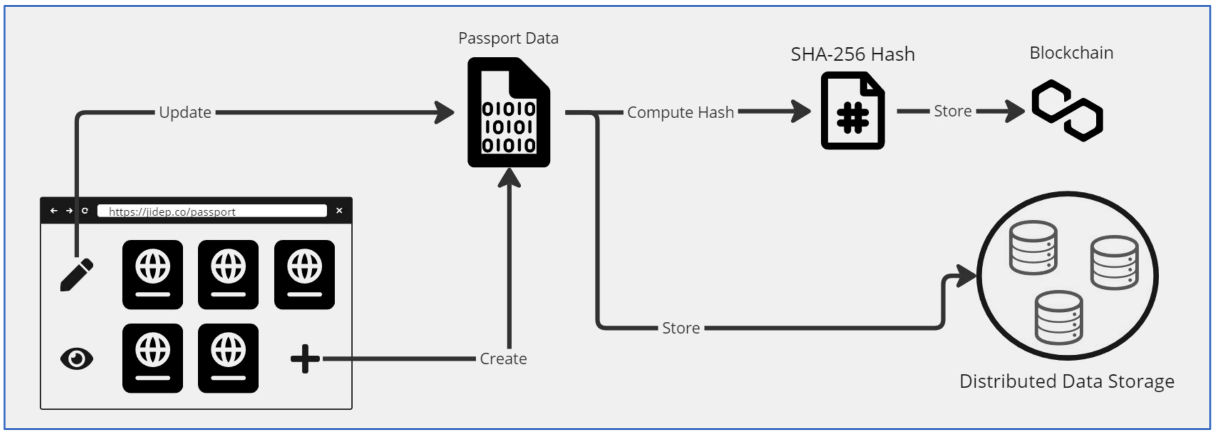


Figure 1 Material Passport data anchoring in DLT

Upon data retrieval from the distributed storage, a hash is recalculated and verified in the blockchain to ensure its integrity, as shown in Figure 2. If any unauthorised data modification occurs, the hash value will change, and we will be notified during the blockchain verification stage.

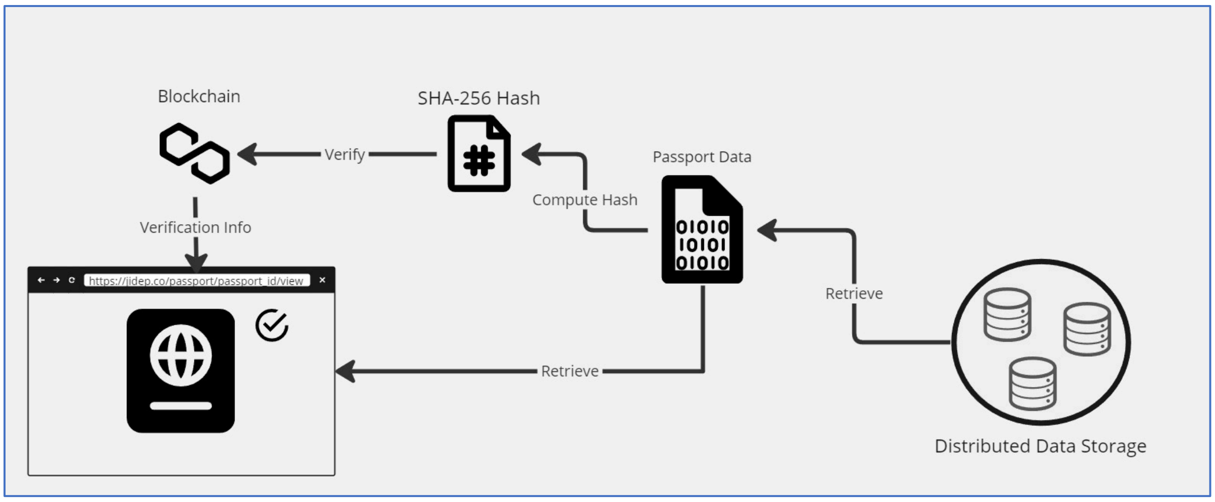


Figure 2 Material Passport data verification with DLT

2.3.4 Hashing

Hashing is a process that involves converting data of varying sizes into a fixed-length string of characters using a mathematical formula known as a cryptographic hash function [12]. This technique is helpful for numerous purposes, such as data authentication, security, storage, and retrieval. Different types of hashing algorithms are available, such as SHA-1, MD5, and SHA-256, each with specific features and applications. We have used the SHA-256 algorithm for the anchoring of our data. This cryptographic hash function converts any data into a fixed-length string of 256 bits. It is part of the SHA-2 family of hash functions, which are designed by the US National Security Agency. SHA-256 is utilised for various purposes, such as data authentication, security, storage, and retrieval. Some of the critical features of SHA-256 that we employed in our application are:

- It provides high security, trust, and immutability, as it is challenging to reverse or modify the output without changing the input. It is also resistant to collisions, so it is unlikely to find two different inputs producing the same output.
- It improves efficiency and transparency, reducing the need for intermediaries and bureaucracy. It can also verify the integrity and validity of the data without revealing the original content.

3. Implementation

3.1 Selecting a language

Smart contracts can be written using many different programming languages [13]. Some of the most used languages for smart contracts include:

- **Solidity:** This object-oriented and statically typed language runs on the Ethereum Virtual Machine (EVM). C++, Python, and JavaScript influence it and support advanced features such as inheritance, libraries, and user-defined types. Solidity is the most widely used language for smart contracts on Ethereum and other blockchains.
- **Rust** is a general-purpose, multi-paradigm language prioritising performance, reliability, and productivity. It can be used to write smart contracts for various blockchains, such as Solana and Polkadot.
- **Vyper:** This is a contract-oriented and Pythonic language that targets the EVM. Vyper is designed to be more secure, simple, and transparent than Solidity and does not support certain features such as modifiers, class inheritance, or inline assembly.

Solidity is our top choice for writing smart contracts due to its compatibility, user-friendliness, thriving community, and extensive usage [13]. Here are some reasons why we picked Solidity:

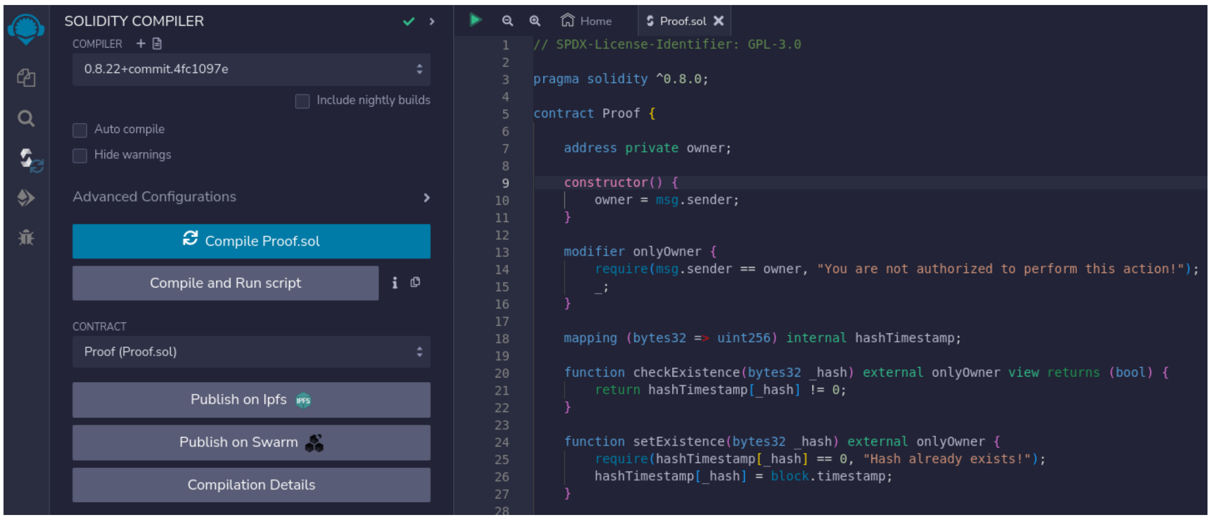
- **Compatibility:** Solidity is specifically designed to run on the Ethereum Virtual Machine (EVM), the backbone of the Ethereum blockchain platform. Polygon, the layer-2 blockchain network, is built on the EVM. The EVM is a distributed computer that carries out smart contracts and transactions on the network. With Solidity, developers can create and deploy smart contracts that interact with the EVM and other smart contracts.
- **Ease of use:** Solidity is a high-level programming language with syntax similar to JavaScript, C++, or Python. It also supports object-oriented features such as inheritance, abstraction, and polymorphism. Solidity simplifies smart contract writing and comprehension for developers and allows them to debug and test their code easily.
- **Community:** Solidity boasts a vast and active community of developers, users, and supporters. Many online resources, such as tutorials, documentation, forums, blogs, podcasts, and videos, can aid developers in learning and improving their Solidity skills. Additionally, many tools, such as compilers, editors, frameworks, libraries, and testing tools, can assist developers in creating and deploying smart contracts.
- **Usage:** Solidity is the most used programming language for smart contracts on the Ethereum blockchain platform. According to a report by ConsenSys, Solidity accounts for 72% of all smart contract code on Ethereum. Many popular applications, such as decentralised finance (DeFi), non-fungible tokens (NFTs), decentralised exchanges

(DEXs), and decentralised autonomous organisations (DAOs), are built using Solidity [14].

3.2 Selecting a development environment

Remix [15] is a web-based Integrated Development Environment (IDE) that allows users to write, test, and deploy smart contracts using Solidity. We chose to use Remix as our IDE for the following reasons:

- As depicted in Figure 3, it enables users to compile smart contracts in the browser using a JavaScript implementation of the Ethereum Virtual Machine (EVM). Users can test their contracts without deploying them to a network.


SOLIDITY COMPILER

COMPILER: 0.8.22+commit.4fc1097e

Include nightly builds

Auto compile

Hide warnings

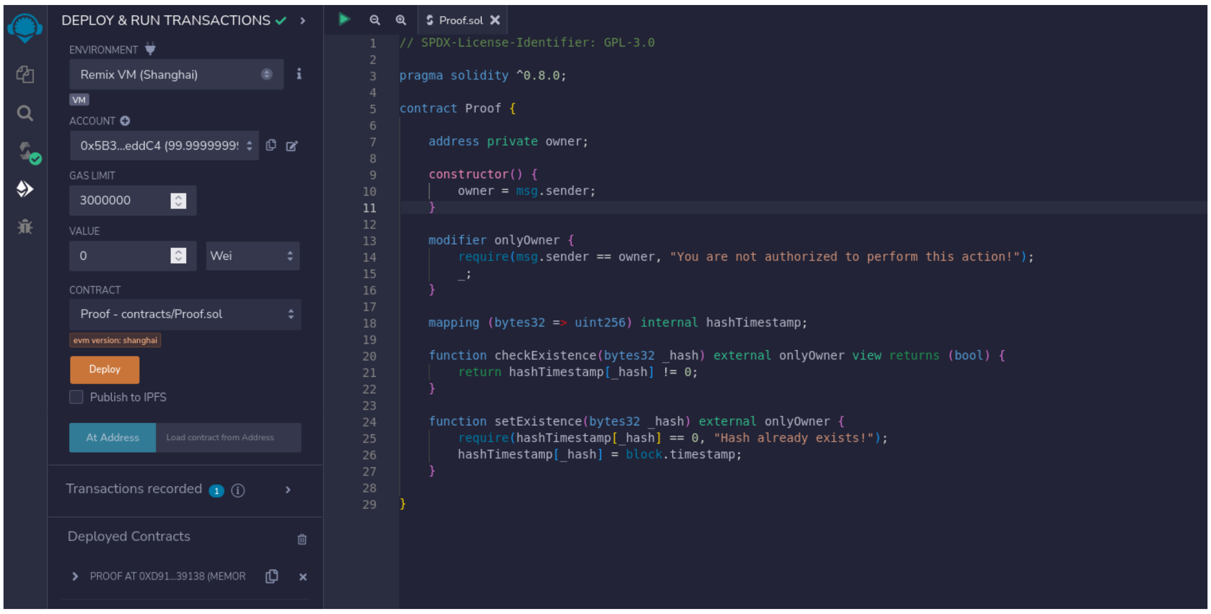
Advanced Configurations

CONTRACT: Proof (Proof.sol)

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.0;

contract Proof {

    address private owner;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner, "You are not authorized to perform this action!");
        _;
    }

    mapping (bytes32 => uint256) internal hashTimestamp;

    function checkExistence(bytes32 _hash) external onlyOwner view returns (bool) {
        return hashTimestamp[_hash] != 0;
    }

    function setExistence(bytes32 _hash) external onlyOwner {
        require(hashTimestamp[_hash] == 0, "Hash already exists!");
        hashTimestamp[_hash] = block.timestamp;
    }
```

Figure 3 Compiling smart contract with Remix IDE.

- As depicted in Figure 4, it allows users to connect to different networks, such as the in-browser JavaScript VM, local nodes, or public networks. This means that users can choose the network to deploy their contracts and use different accounts and wallets to manage their transactions.


DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Remix VM (Shanghai)

VM

ACCOUNT: 0x5B3...eddC4 (99.9999999!

GAS LIMIT: 3000000

VALUE: 0 Wei

CONTRACT: Proof - contracts/Proof.sol

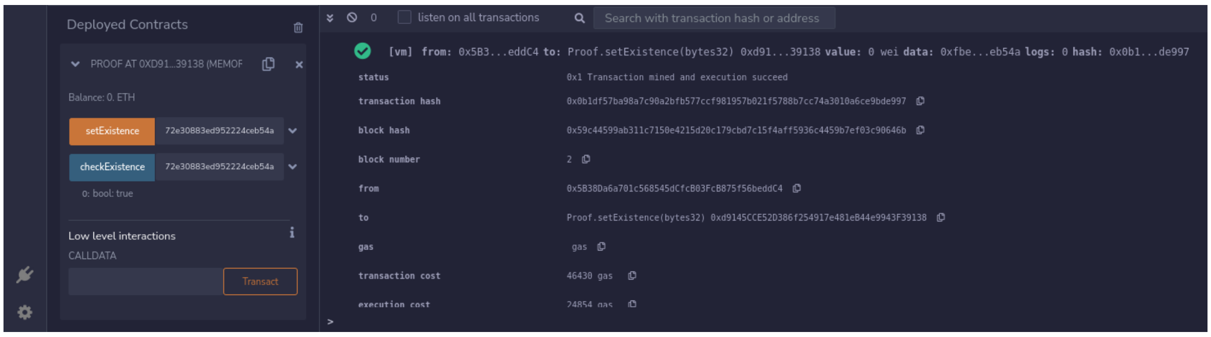
evm version: shanghai

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.0;

contract Proof {

    address private owner;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner, "You are not authorized to perform this action!");
        _;
    }

    mapping (bytes32 => uint256) internal hashTimestamp;

    function checkExistence(bytes32 _hash) external onlyOwner view returns (bool) {
        return hashTimestamp[_hash] != 0;
    }

    function setExistence(bytes32 _hash) external onlyOwner {
        require(hashTimestamp[_hash] == 0, "Hash already exists!");
        hashTimestamp[_hash] = block.timestamp;
    }

}
```

Figure 4 Deploying smart contract with Remix IDE.

European Commission

- Figure 5 depicts a user-friendly interface that allows users to interact with their contracts, view their transactions, and debug their code. Additionally, it offers various tools and plugins that can help users with code analysis, unit testing, documentation, and more.

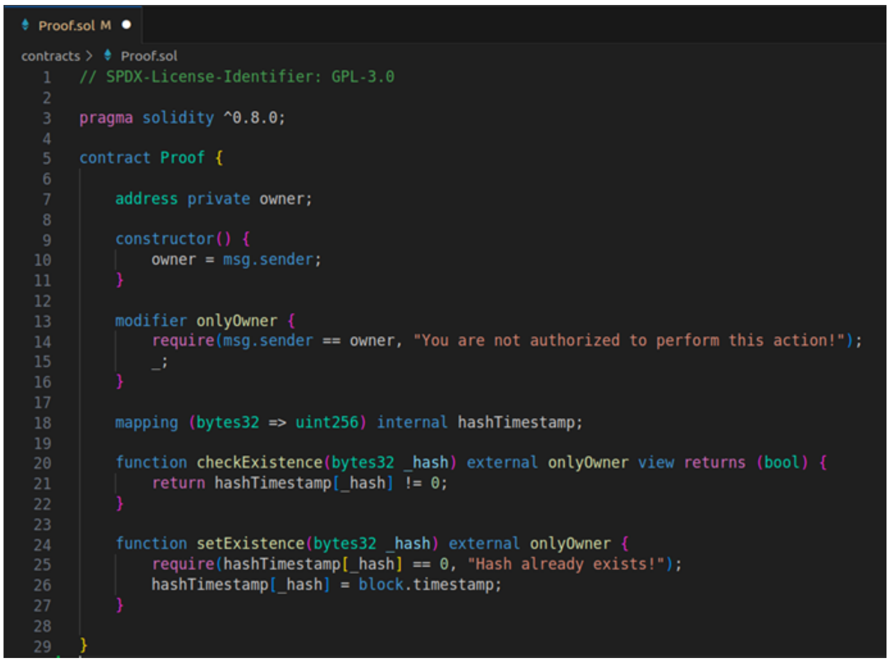


```
[vm] from: 0x5B3...eddC4 to: Proof.setExistence(bytes32) 0xd91...39138 value: 0 wei data: 0xfbe...eb54a logs: 0 hash: 0x0b1...de997
status                    0x1 Transaction mined and execution succeed
transaction hash          0x0b1df57ba98a7c90a2bfb577ccf981957b021f5788b7cc74a3010a6ce9bde997
block hash                0x59c44599ab311c7150e4215d20c179cbd7c15f4aff5936c4459b7ef03c90646b
block number              2
from                      0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                        Proof.setExistence(bytes32) 0xd9145CCE52D386f254917e481eB44e9943F39138
gas                        gas
transaction cost          46430 gas
execution cost            24854 gas
>
```

Figure 5 Interacting and testing the smart contract with Remix IDE.

- It is compatible with Ethereum and other EVM-compatible blockchains such as Polygon, Binance Smart Chain, or Avalanche. This allows users to write and deploy smart contracts interacting with these blockchains and their applications.

3.3 Developing smart contract



```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.0;

contract Proof {

    address private owner;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner, "You are not authorized to perform this action!");
        _;
    }

    mapping (bytes32 => uint256) internal hashTimestamp;

    function checkExistence(bytes32 _hash) external onlyOwner view returns (bool) {
        return hashTimestamp[_hash] != 0;
    }

    function setExistence(bytes32 _hash) external onlyOwner {
        require(hashTimestamp[_hash] == 0, "Hash already exists!");
        hashTimestamp[_hash] = block.timestamp;
    }

}
```

Figure 6 Smart Contract for anchoring data in DLT.

Figure 6 depicts the smart contract developed to anchor data in DLT. It operates in the following steps:

1. When the smart contract is deployed on an EVM-compatible blockchain, the user's address that initiated the deployment is marked as the owner. In the case of the JIDEP platform, it owns the contract.

2. The smart contract owner can then store a single hash at a time in a mapping data structure, a tool of the Solidity programming language that allows the user to store data as key-value pairs. In our case, the keys are the hashes, and the values are the timestamps of their respective blockchain transactions.

3. The smart contract checks for the existence of a given hash and returns a Boolean value. If the hash is present in the stored mapping, the function returns true. Otherwise, it returns false.

3.4 Continuous improvement

Gas is spent on every storage operation on the blockchain. Gas is like the currency of blockchain transactions, and it incurs a cost. So, keeping in mind that the operation of the JIDEP project will be scaled, we have improved our smart contract to reduce costs and make our solution more scalable. The improved version of the smart contract allows us to store multiple hashes at once, reducing the gas cost by almost 50%.

3.5 Testing the smart contract

Testing smart contracts is the process of verifying that the code of a smart contract works as expected. Testing smart contracts is essential for ensuring the security, reliability, and usability of smart contracts and preventing errors, bugs, or exploits that may cause losses or damages. Testing smart contracts can help developers improve code quality, avoid costly mistakes, and deliver better products and services. Testing smart contracts requires understanding the business logic and workflow of the smart contract, evaluating the assumptions and expectations related to the contract execution, and using well-developed testing frameworks and tools to check the functionality and performance of the contract [16]. Testing smart contracts can be done using various methods, such as:

- **Manual Testing:** This method involves manually inspecting and interacting with the code of a smart contract. Manual testing can use tools like Remix, Etherscan, or Ganache to compile, deploy, and debug smart contracts on different networks to analyse the code for vulnerabilities, such as reentrancy, overflow, or underflow. We have used Remix, Ganache, and Hardhat's networks for testing. We also used Polygonscan, which shows us the transaction details of our interaction with the Polygon Testnet.

- **Automatic Testing:** This method uses tools that automatically check a smart contract's code for errors in execution. Automated testing can use frameworks like Hardhat, Truffle, or Waffle to write and test cases that simulate different scenarios and inputs. We have used the Hardhat framework with additional testing libraries to automate our smart contract testing.

The details of our testing activities will be discussed in the deliverable report D4.3.

4. Integration

4.1 Choosing framework

Integrating with the right tools is crucial when working with any technology. In the case of blockchain, several frameworks can be used. Here, we'll compare two of the top frameworks for EVM-compatible blockchain integration: Hardhat and Truffle.

Hardhat [17] is a highly customisable Ethereum development environment that supports smart contract debugging, testing, and deployment. It comes with an Ethereum network simulator that offers advanced features such as console logging, stack traces, error messages, code coverage, and debugging. Hardhat supports multiple programming languages, such as Solidity and JavaScript. It has a rich ecosystem of plugins that extend its functionality and integrate with popular tools such as Ethers.js, Waffle, Truffle, TypeChain, Solhint, and more.

On the other hand, Truffle [18] is one of the world's most widely used Ethereum development frameworks. It provides a comprehensive suite of smart contract development, testing, and deployment tools, including smart contract debugging, migrations, network management, and interactions. Truffle supports Solidity and Vyper as programming languages and has a large developer community and a wealth of resources such as tutorials, documentation, and plugins.

Of the two, we chose to work with the Hardhat framework due to its extreme flexibility. It allows customisation and modification of anything from configuration to tasks and the ability to extend its functionality with plugins or write its own plugins. Hardhat also enables deeper interoperability and integration with existing tools, such as Truffle, Waffle, Ethers.js, Web3.js, TypeChain, Solhint, and more, while providing complete control over the development environment and workflow.

Furthermore, Hardhat comes with its own Ethereum network simulator, the Hardhat Network, which runs on your machine and provides advanced features such as console logging, stack traces, error messages, code coverage, and debugging. You can also use Hardhat Network to fork the mainnet or any other network and test your contracts with actual data. Additionally, Hardhat uses its own Ethereum Virtual Machine (EVM) implementation that is faster and more reliable than other EVMs and supports Solidity 0.8.x features such as checked arithmetic and custom errors, making it easier to compile, deploy, and test your smart contracts.

4.2 Choosing library

Ethers.js [19] and Web3.js [20] are popular JavaScript libraries that enable frontend applications to interact with the Ethereum blockchain through deployed smart contracts. Although they have similar functionality, they also have some key differences.

Ethers.js is a lightweight and compact library that aims to provide a complete and straightforward solution for Ethereum development. It supports multiple programming languages, such as JavaScript and TypeScript. Additionally, Ethers.js supports ENS names, which can be used anywhere an Ethereum address can be used. One of Ethers.js's key advantages is its better support for wallets and authentication, as it allows you to create and manage multiple wallets and signers. Furthermore, Ethers.js has a rich ecosystem of plugins that extend its functionality and integrate with popular tools such as Waffle, Hardhat, TypeChain, and more.

Web3.js, on the other hand, is the first and most widely used library for Ethereum development. It supports Solidity and Vyper as programming languages. Web3.js also supports multiple networks, such as Ethereum mainnet, testnets, private networks, and sidechains. Web3.js is more versatile and flexible, allowing you to customise and modify anything you like, from the configuration to the tasks. Moreover, it has a large community of developers and various resources such as tutorials, documentation, and plugins.

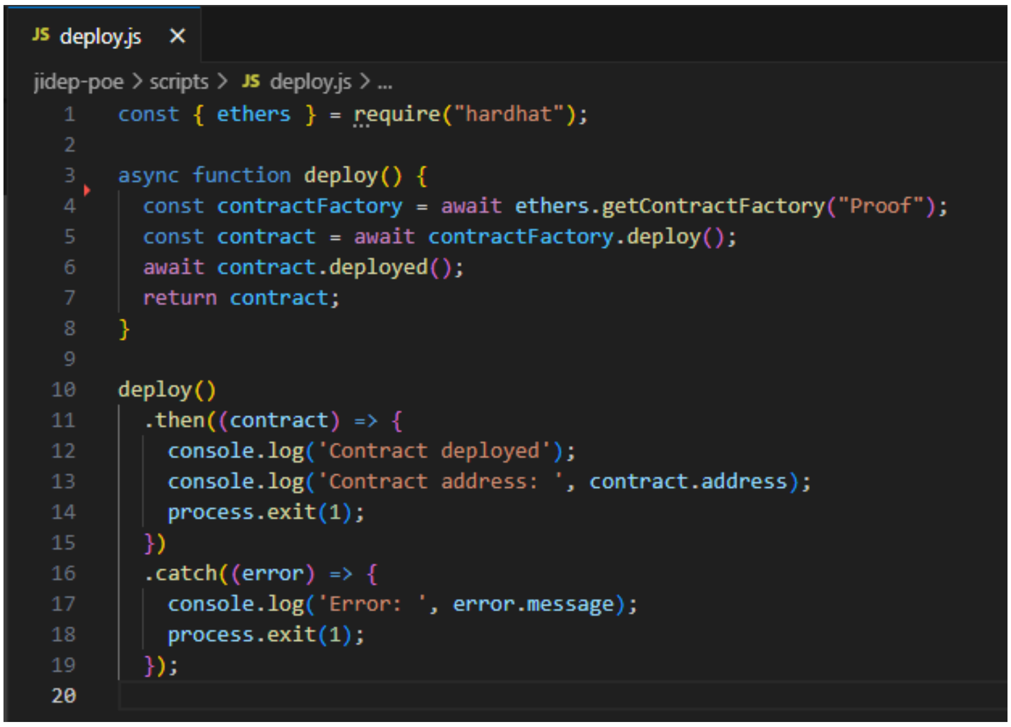
For our script, we chose to work with Ethers.js. The main feature of Ethers.js, which became the deciding factor for choosing, is that it separates the API into two distinct roles: the provider, which is an anonymous connection to the Ethereum network, and the signer, which has access to the private key and can sign transactions. This separation of concerns provides more flexibility and security to developers. In contrast, Web3.js provides a single instantiated web3 object with methods for interacting with the blockchain. Furthermore, Ethers.js is easily integrable with our chosen framework, Hardhat.

Since both libraries are constantly being developed and improved, the necessity to switch libraries may arise at some point. However, we have yet to feel the need to change libraries.

4.3 Writing integration script

Using scripts to deploy and interact with smart contracts provides us with some advantages, such as:

- It can improve development efficiency and productivity by automating and streamlining the processes of creating and using smart contracts.
- It can enhance the functionality and performance of applications by enabling more complex and dynamic interactions with smart contracts.
- It can facilitate the testing and debugging of smart contracts by allowing developers to simulate different scenarios and inputs and to monitor and analyse the results.


deploy.js

jidep-poe > scripts > deploy.js > ...

```javascript
const { ethers } = require("hardhat");

async function deploy() {
  const contractFactory = await ethers.getContractFactory("Proof");
  const contract = await contractFactory.deploy();
  await contract.deployed();
  return contract;
}

deploy()
  .then((contract) => {
    console.log('Contract deployed');
    console.log('Contract address: ', contract.address);
    process.exit(1);
  })
  .catch((error) => {
    console.log('Error: ', error.message);
    process.exit(1);
  });
```

Figure 7 Script to deploy the smart contract

Deployment using the script: When it comes to deploying smart contracts through scripts, it involves utilising a tool or framework that is specifically designed to compile, deploy, and manage smart contracts on a blockchain network. For instance, Truffle and Hardhat are two of the most popular tools for deploying smart contracts on Ethereum and other EVM-compatible blockchains. With these tools, developers can write and execute scripts to deploy smart contracts to various networks, such as local, test, or main networks. In this particular case, we have utilised the Hardhat framework to deploy our smart contracts depicted in Figure 7.

Interacting through the script: Interaction with smart contracts through scripts involves using a tool or library to communicate and interact with smart contracts on a blockchain network. For example, Web3.js and Ethers.js are popular libraries for interacting with smart contracts on Ethereum and other EVM-compatible blockchains. These libraries allow developers to write and run scripts that can call functions, read data, send transactions, and listen to events from smart contracts. We developed the integration script using the ethers library to call the smart contract methods. These deployment and integration scripts were then used by the Proof-of-Existence (PoE) API from T3.5 (Implementation of JIDEP APIs and SDK) to integrate with the DLT functionalities. We can store and check the existence of hashes that we receive through our PoE API.

4.4 Continuous improvement

Blockchain development is a fast-changing ecosystem, so the frameworks and libraries are updated frequently. To make our scripts compatible, we must upgrade and make necessary changes to utilise the new features provided by the updated versions. We will do this occasionally to ensure that our script is up-to-date.

5. DLT platform enabled secondary features

5.1 Access and permissions framework

JIDEP’s primary function is to facilitate industrial data exchange. Users' permissions to search, read, and modify data within the distributed storage system will be determined by an access and permission control framework, which will do so after validating the users' credentials. The framework will also make it possible for data owners' access management policies to be more flexible and granular. Data owners may provide access and permissions following stakeholders' requirements and consent. Similar to the data to be exchanged between stakeholders, the access and permission-related data will also be anchored in blockchain to ensure that the data transactions are eliminated from fraud. To guarantee that JIDEP is a secure, morally and legally sound system, it will embrace Security and Privacy by Design principles, beginning with the requirements and architecture. This will involve integrating security, privacy, and data protection into the design and development of the components and the integrated implementation of JIDEP. The access and permission control framework for the JIDEP platform stores the cryptographic hash of access control information and only allows access after successful verification from our DLT platform.

5.2 Audit trail framework

Maintaining a record of activities or changes affecting data is the primary purpose of an audit trail framework. It is a systematic approach to track and record such activities to ensure complete, comprehensive, and immutable records. It will help verify the security and integrity of the material passport data, assuring accountability and aid in reconstructing events. The audit trail framework will serve two primary purposes: data provenance and usage tracking.

Data provenance involves maintaining a record of the origin of data and the path it takes over time. This includes information such as who created the data, when it was created, and its various transformations. The goal is to ensure that data has not been tampered with and is still reliable.

Data usage tracking involves tracking who accessed or modified the data, what changes were made, and when these activities occurred. This information is necessary to comply with various regulations, such as GDPR or HIPAA, which require detailed data access and usage records.

The audit trail framework will accurately record all data usage and provenance activities. Every activity on the platform generates a new hash of data, thus creating a trail of hashes that can be traced back to the activity that generated the hash.

5.3 Governance framework

Formulating a governance framework for a platform leveraging a public blockchain like Polygon for data anchoring, with data residing in decentralised off-chain storage, involves multiple critical steps to be undertaken:

- **Selection of Off-Chain Storage Technology:** When selecting off-chain storage technology, it is essential to prioritise its reliability, security, and economic sustainability. Considerations should include the technology's track record with confidential data, system uptime, and distribution across various regions. Our chosen distributed off-chain storage option is summarised in deliverable report D2.7.
- **Maintenance of Off-Chain Storage Protocols:** Our governance plan incorporates a protocol for the routine upgrade and upkeep of off-chain storage systems. This could entail adopting new technological advances or enhancing existing infrastructures to boost security or operational efficiency. Our system architecture adopts a microservice

approach for off-chain storage, as outlined in deliverable report D1.3, facilitating straightforward updates or technology shifts.

- **Regulation of Access and Permissions:** Access to off-chain data will be tightly regulated via a comprehensive permissions framework. This framework will assign read, write, and modification rights based on defined roles for various stakeholders, including users, administrators, and auditors.
- **Security, Privacy, and Regulatory Compliance:** The platform uses advanced encryption to secure data during storage and transmission. We will conduct periodic security audits and tests to uncover and address vulnerabilities. Compliance with privacy laws like GDPR, HIPAA, and others pertinent to our user demographic is mandatory, encompassing policies on data retention, deletion rights, and data mobility. The governance framework is designed to be agile to accommodate new legal requirements.
- **Smart Contract Policy Enforcement:** Smart contracts on the Polygon blockchain will be deployed to systematise elements of the governance model. This will support the automated processing of data anchoring tasks.
- **Audit Trails and Transparency Reports:** The system will be engineered to comprehensively log all blockchain and off-chain storage activities, supporting audit requirements.

These pillars will form the foundation of an effective governance strategy, ensuring that the platform remains secure, efficient, and within regulatory bounds.

6. Conclusions

Integrating Distributed Ledger Technology (DLT) into the JIDEP platform is important in promoting efficient industrial data exchanges and endorsing a sustainable circular economy. The choice of the Ethereum-based Polygon network as the backbone for DLT features was informed by thoroughly considering critical factors such as scalability, interoperability, security, and the network's environmental footprint. This strategic decision underscores JIDEP's commitment to a forward-thinking, resilient technological infrastructure.

During the development phase, JIDEP focused on building a robust, secure, and user-friendly system. By utilizing Solidity, Remix, and Hardhat, they ensure effective implementation of smart contracts for data anchoring, fostering trust and reliability through a commitment to data integrity and authenticity across the platform's ecosystem.

Transitioning to a public blockchain, specifically the Polygon network, JIDEP is adopting the inherent benefits of public blockchains. These benefits, including decentralisation, transparency, and cost efficiency, not only position JIDEP at the forefront of technological advancement but also paint a promising picture for potential investors. As JIDEP's platform evolves, it continues to refine its governance framework and DLT components to meet stakeholder needs and align with changing industrial trends, ensuring it remains a dynamic and responsive entity within the industry.

References

[1] "Blockchain and Distributed Ledger Technology (DLT)," [Online]. Available: https://www.worldbank.org/en/topic/financialsector/brief/blockchain-dlt. [Accessed October 2023].

[2] "Polygon PoS | The most efficient blockchain protocol," [Online]. Available: https://polygon.technology/polygon-pos. [Accessed October 2023].

[3] "What Are Smart Contracts on the Blockchain and How They Work," [Online]. Available: https://www.investopedia.com/terms/s/smart-contracts.asp. [Accessed November 2023].

[4] "How Blockchain Can Improve Data Security and Integrity," [Online]. Available: https://zaisan.io/blockchain-data-security-integrity. [Accessed October 2023].

[5] "Here's how we can turn more industries into circular economies," [Online]. Available: https://www.weforum.org/agenda/2023/01/industry-circular-economy-change. [Accessed October 2023].

[6] "Responsible industry | UNEP - UN Environment Programme," [Online]. Available: https://www.unep.org/explore-topics/resource-efficiency/what-we-do/responsible-industry. [Accessed October 2023].

[7] "Components of Blockchain: Explained," [Online]. Available: https://www.staderlabs.com/blog/blockchain-components. [Accessed October 2023].

[8] "How To Pick The Best Consensus Algorithm For Blockchain?," [Online]. Available: https://blockchain-council.org/blockchain/how-to-pick-the-best-consensus-algorithm-for-blockchain. [Accessed October 2023].

[9] "What Is Proof of Work (PoW) in Blockchain?," [Online]. Available: https://www.investopedia.com/terms/p/proof-work.asp. [Accessed October 2023].

[10] "Proof of stake - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Proof_of_stake. [Accessed October 2023].

[11] "Blockchain Anchor – Blockchain Patterns," [Online]. Available: https://research.csiro.au/blockchainpatterns/general-patterns/self-sovereign-identity-patterns/anchoring-to-blockchain. [Accessed October 2023].

[12] "What Is Hashing, and How Does It Work?," [Online]. Available: https://www.codecademy.com/resources/blog/what-is-hashing. [Accessed October 2023].

[13] "Smart contract languages | ethereum.org," [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/languages. [Accessed November 2023].

[14] "Ethereum by the Numbers – June 2020 | Consensys," [Online]. Available: https://consensys.io/blog/ethereum-by-the-numbers-june-2020. [Accessed November 2023].

[15] "Remix - Ethereum IDE & community," [Online]. Available: https://remix-project.org. [Accessed November 2023].

[16] "Testing smart contracts | ethereum.org," [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/testing/. [Accessed November 2023].

[17] "Hardhat | Ethereum development environment," [Online]. Available: https://hardhat.org. [Accessed November 2023].

[18] "Home - Truffle Suite," [Online]. Available: https://trufflesuite.com. [Accessed November 2023].

[19] “ethers-io/ethers.js: Complete Ethereum library and wallet implementation in JavaScript.,” [Online]. Available: https://github.com/ethers-io/ethers.js. [Accessed November 2023].

[20] “web3.js - Ethereum JavaScript API,” [Online]. Available: https://web3js.readthedocs.io/en/v1.10.0/. [Accessed November 2023].