

# Extending the Docstone to Enable a Blockchain-based Service for Customizable Assets and Blockchain Types

Pamella Soares □ [ State University of Ceara | [pamella.soares@aluno.uece.br](mailto:pamella.soares@aluno.uece.br) ] Raphael Saraiva □ [ State University of Ceara | [raphael.saraiva@aluno.uece.br](mailto:raphael.saraiva@aluno.uece.br) ] Iago Fernandes □ [ State University of Ceara | [iago.fernandes@aluno.uece.br](mailto:iago.fernandes@aluno.uece.br) ] Allysson Alex Araújo □ [ Federal University of Ceara | [allysson.araujo@crateus.ufc.br](mailto:allysson.araujo@crateus.ufc.br) ] Jerffeson Souza □ [ State University of Ceara | [jerffeson.souza@uece.br](mailto:jerffeson.souza@uece.br) ] Ricardo Loiola □ [ Pontifical Catholic University of Rio Grande do Sul | [ricardo.loiola@edu.pucrs.br](mailto:ricardo.loiola@edu.pucrs.br) ]

## Abstract

Document management ensures proper document production, storage, and use in many organizations. The digitization process has become a relevant factor for improving document management processes in several aspects. In this context, blockchain is relevant due to its ability to meet the requirements of any document management solution. However, there are still difficulties in the efficient implementation of blockchain infrastructure. On the other hand, existing solutions propose specific context proposals and strict business rules. Therefore, this paper extends previous studies that propose an architecture designed to provide and facilitate the integration of blockchain-based assets for registration and verification for third-party applications from different domains called Docstone. This solution allows configuring parameters to create customizable asset templates and selecting blockchains through a developer-friendly Application Programming Interface. The improvements provided a discussion of an adapted software architecture of Docstone to support new assets and smart contracts in a personalized manner and a mechanism that automatically generates IPFS directories mapped to each smart contract. We conducted an assessment by performing smart contracts deployment and data write and read operations using different asset types, smart contract codes, and blockchains to evaluate its behavior in Docstone. We also analyzed design decisions for decentralized applications and evaluated the performance of different assets and blockchains for information storage. This study revealed that NFTs have higher deployment latency times than documents in Sepolia and Alfajores. However, documents have the lowest latency time in both networks for writing and reading requests. Alfajores was the blockchain that stood out the most with the least latency of operations.

**Keywords:** Document Management, Blockchain-as-a-Service, Application Programming Interface

## 1 Introduction

A document represents a set of data capable of being used for queries, evidence, and research so that facts and thoughts of a person can be verified at a given time (de Oliveira Melo and Neto, 2014). Due to the increase in the document volume of organizations, resources, and processes are required to properly manage all information, including guaranteeing the production, storage, and correct use of documents from public and/or private institutions (Morais et al., 2021).

Information technology has become essential in supporting activities related to document management, as it can improve the efficiency of services, document management, reduction and optimization of storage space, and sharing information, in addition to mitigating the loss of records (Ab Aziz et al., 2020). With document management systems, it is possible to eliminate paper and use “digital documents”, represented by information in media, text, or images in electronic format stored and manipulated by a computer (Kim, 2020). However, when managing documents, organizations must also ensure both privacy and information security for sensitive and

confidential data, as well as transparency regarding the availability of public information.

In light of the previous arguments, blockchain technology assumes a prominent role in the storage process, given its potential to guarantee the requirements of integrity, authenticity, access control, transparency, and availability necessary for document management solutions. In summary, blockchain combines cryptography, data management, networking, and incentive mechanisms to support the verification, execution, and recording of transactions between different parties without the central control of any trusted third party (Xu et al., 2019a). In this sense, the nodes insert information into the blockchain by agreeing through consensus algorithms responsible for ensuring the integrity of the shared data.

Although the promising results from using blockchain for information management and storage, barriers to its development and adoption can be found (Wan et al., 2018). In particular, the implementation process of blockchain and decentralized applications (dApps) can become complex and prone to mistakes since one must consider network infrastructure components and Smart Contract (SC) business rules in the most varied aspects of a software architecture (Jie et al., 2021). Furthermore, as the blockchain is essentially a

distributed system, the deployment challenge and the high cost of operating and maintaining the blockchain jeopardize the curve of adoption in the market (Li et al., 2021).

One current path to disseminating the blockchain's adoption is using "Blockchain-as-a-Service" (BaaS) platforms. In brief, a BaaS enables developers to focus only on coding business rules since it provides cloud services such as infrastructure deployment and network monitoring (Onik and Miraz, 2019). However, despite promoting improvements in developer productivity and being powerful tools for developing decentralized applications, the use of BaaS, such as *Amazon Managed Blockchain*, *Huawei Blockchain Service*, and *Oracle Cloud Infrastructure Blockchain Platform*, still demand considerable study in their extensive documentation to understand how they work, as each one of them has unique features and tools (Jie et al., 2021). These processes are only sometimes necessary for simple systems, such as recording and validating document information.

On the other hand, if we consider the context in which the literature presents the current proposals, we can highlight the constraints that arise from the particular specificity of domains and rigid business rules. For example, there are specific platforms based on blockchain for recording certificates and diplomas (Saraiva et al., 2021; Abreu et al., 2020), recording medical information (Afrianto and Heryanto, 2020) or even proposals for recording water dam inspections (Macedo et al., 2021). Nevertheless, these approaches have three common basic requirements: registration, search, and document data validation, which a single solution, generic and customizable architecture could aggregate. Aiming to tackle this opportunity, we have been developing a *Blockchain-based Customizable Asset Registration Service* in the form of a solution called Docstone. In summary, Docstone allows the storage of document information in a personalized and flexible way, from the initial configuration of parameters for creating document templates to its storage and validation in one or more blockchains.

Therefore, this paper is part of a large project and extends two previous articles (Soares et al., 2022a,b), the last one awarded at the 16th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS 2023). A threefold improvement grounds this extension. First, our analysis expanded to include new asset types beyond Documents (as in Docstone's previous version), such as Processes and NFTs. Including new asset types allowed us to investigate how different SC types' performances differ. In addition, we can explore the use of NFTs for tokenizing documents for future commercializing on blockchains. Second, we integrated distributed file storage capabilities into our solution and two new blockchains (Sepolia and Alfajores). By doing so, we evaluated the performance of asset types when using file media in several formats beyond the traditional metadata, which is typically limited to text and numerical data. Lastly, we conducted a comprehensive analysis comparing the performance of Processes, Documents, and NFTs, including different blockchain

networks. By exploring these perspectives, we were able to provide a more nuanced and comprehensive understanding of the performance of different assets (and SC types) considering different blockchain networks.

Ultimately, the relevance of this paper relies on the following contributions: 1) clarifying the software architecture adaptation of the Docstone to support new assets and smart contracts in a generic and customized way to cover other use cases (such as process tracking and tokenization of assets through NFTs); 2) discussing a mechanism for automatically creating IPFS directories mapped to each smart contract for organized storage of the several media formats related to assets, 3) providing a comprehensive analysis of design decisions for decentralized applications, and 4) detailing an empirical assessment of assets type to compare the execution performance of different smart contract codes and blockchains.

The work is organized into eight sections, including this introduction. Section 2 details the conceptual background. Section 3 approaches the related work. Section 4 covers the architecture of the proposed solution. Section 5 provides a brief demonstration of the designed software architecture. Section 6 elucidates the empirical study and analysis. Section 7 clarifies the threats to the validity of this study. Finally, Section 8 highlights the final remarks.

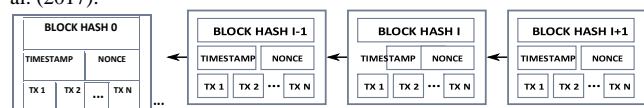
## 2 Background

This section provides an overview of the fundamental concepts related to blockchain, including its structure and operation. It covers the two main types of blockchains (public and private) and provides examples of the technologies and standards used for implementing smart contracts. Additionally, we briefly describe the InterPlanetary File System.

### 2.1 Blockchain

Blockchain is a peer-to-peer (P2P) network that employs a distributed ledger to store transactions through network consensus. Due to its use of cryptography, smart contracts, and other components, blockchain becomes a secure and reliable network (Acharya et al., 2019). In summary, a blockchain is a chain of blocks composed of transactions stored sequentially. Each newly validated block is interconnected with the last block added to the network through a cryptographic hash (Van Mölken, 2018). This feature enables the integrity and immutability of information, as shown in Figure 1.

**Figure 1.** Simplified representation of a blockchain. Adapted from Beck et al. (2017).



GENESIS BLOCK      BLOCK I      BLOCK I+1      BLOCK I+2

Indeed, blockchain was introduced with Bitcoin, an electronic currency transacted on a P2P network without intermediaries to transfer payments between peers (Bashir, 2017). Among the practical elements of this solution, one can highlight the possibility of consensus in an open network with unknown participants, the guarantee of auditability, authenticity, availability, non-repudiation, and integrity of transactions validated and stored in the distributed ledger (Greve et al., 2018). Thus, blockchain has been integrated into various domains of applications with different implementation particularities.

To meet these demands, a blockchain can be categorized into public or private networks (Van Mölken, 2018). In public networks, any node can join and write and read transactions. Furthermore, the nodes are anonymous, as there is no need to verify the identity of network participants (such as Ethereum). Regarding private (also known as permissioned networks), only previously authorized users or organizations can perform writing and/or reading operations, as in the blockchains developed with Hyperledger Fabric.

Xu et al. (2019b) clarified the relationship between blockchain types and their respective permission types. Both permissionless and permissioned public networks offer transparency, disintermediation, and anonymity (Acharya et al., 2019). The difference is that everyone in the nonpermissioned public network can join the blockchain, forming a network of unreliable parties. All nodes can read, write, and validate transactions. In public permissioned networks, all users can read transactions, but only some have permission to write or vice versa. In this case, there are restrictions on reading and/or writing. On private and non-permissioned blockchain networks, only selected members can perform actions such as recording, reading, and validating transactions. In general, permissions are centralized to certain members, which may even eliminate the decentralized nature of the network but favor possible privacy needs. In private permissioned blockchains, the nodes are identifiable, the permissions are managed, and the privacy of the transacted data can be exclusive to a predefined group of participants, as well as the control of the network consensus.

### 2.1.1 Blockchains Compatible with Ethereum Virtual Machine (EVM)

After Bitcoin's deployment, new generations of blockchains allowed the development of other business logic and models, expanding the use of blockchain beyond the finance domain. In particular, this capability was achieved through Smart Contracts (SCs) introduced by the Ethereum network (Buterin et al., 2013). In short, SCs consist of code that autoexecutes business logic when certain programmed conditions are met (Bashir, 2017). The Ethereum network can be interpreted as a transaction-based state machine in which P2P network nodes have a shared view of a global state. When transactions are executed, a

particular state passes to some final state, representing the Ethereum network's current state. When interacting with the network, the user issues a transaction representing a valid state transition. Then, the nodes choose a set of transactions not yet confirmed from the mempool to verify their validity, perform the appropriate computation, and update the state (Tikhomirov, 2017).

Before submitting transactions permanently included in the blockchain, network nodes called "miners" (in the case of Ethereum 1.0) need to participate in a consensus protocol. One of the best-known consensus protocols is Proof-of-Work (PoW), based on the idea that a random node is selected to create a new block of transactions. A mining node can prove that it has performed a non-trivial computational calculation by solving a cryptographic challenge through brute force (Greve et al., 2018). The node that finds the value of nonce will transmit the block to other nodes that should mutually verify the value of the hash. If the block is validated, other miners will attach it to the blockchain. The miner who discovered it would receive a reward (Zheng et al., 2017).

The shared global state comprises two types of accounts that can interact with each other: Externally Owned Accounts (EOA), controlled by public-private key pairs, and Smart Contract Accounts, controlled by stored code along with the account (Guide, 2021). According to Tikhomirov (2017), the account's status is defined by the following fields: (i) the nonce, which, in the case of EOA, will be the number of transactions sent by that account or, in the case of SC accounts, will be the number of SC creations made by this account; (ii) balance, which is the number of weis belonging to this address; (iii) storageRoot, the Merkle tree root of this account's storage; and (iv) codeHash, which is the hash of this account's contract bytecode.

The Ethereum network allows computer programs to be written in a Turing-complete programming language. An SC is self-executed on each network node by invoking transactions. In this way, the blockchain works as a distributed Virtual Machine (VM) since each node runs a particular VM (Christidis and Devetsikiotis, 2016). On Ethereum, SC code is written in a stack-based bytecode language and executed on the Ethereum Virtual Machine (EVM). The EVM has memory, where items are stored as word-addressed byte arrays, and is volatile as it is not permanent. In addition, the EVM also has non-volatile storage maintained as part of the system state. It should be noted that the EVM stores the program code in a separate virtual read-only memory, which can only be accessed through special instructions (Guide, 2021).

Smart contracts can automate and manage the execution of legal contracts between different parties based on previously defined interaction protocols. To maintain a specification of smart contract standards, mainly related to the development of tokens and their management, Ethereum Request for Comments (ERCs) are created. ERCs are technical standards for Ethereum (or EVM) based tokens published and shared as

Ethereum Improvement Proposals (EIPs)<sup>1</sup>, which “describe standards for the Ethereum platform, including core protocol specifications, client APIs, and contract standards”. EIPs comprise core protocol specifications, covering those already implemented and released or those planned to be, along with client APIs and contract standards.

There are different types of ERCs, most notably the ERC20<sup>2</sup> and the ERC-721<sup>3</sup>. For example, Figure 2 presents standards’ Unified Modeling Language (UML) class diagram.

**Figure 2.** UML classes diagrams representing ERC-20 and ERC-721. Adapted from Stefanović et al. (2022)



The ERC-20 represents fungible tokens (FTs), which are different amounts of identical (fungible) assets and are often employed for implementing cryptocurrencies for EVM-based networks. In turn, ERC-721 represents non-fungible tokens (NFTs), which can represent ownership of many different types of assets, such as real estate or unique works of art. The ERC-721 standard provides basic functionality for tracking and transferring NFTs, considering when tokens are transacted by their owners or by consigned third parties.

In order to develop improvements for the Ethereum network, blockchains compatible with the EVM have been widely proposed. In addition to Ethereum and Ethereum Classic, our work explores the following networks: 1) BNB Smart Chain, 2) Polygon, 3) Klaytn and 4) Celo. These networks are briefly described below.

**BNB Smart Chain** supports consensus layers and hubs for multi-chains and cross-chains. As one of the components of the BNB Chain ecosystem, it is a solution whose initial objective was to provide programmability and interoperability to Beacon Chain, another blockchain developed by Binance and its community that implements a decentralized exchange for digital assets. BNB Smart Chain has a system containing 21 active validators using the Proof of Stake Authority (PoSA) consensus. This type of consensus enables short lock times and lower rates (BNBChain, 2022).

**Polygon** is a scaling solution for public off-chain blockchains composed of a decentralized network of Proof-of-Stake (PoS) validators (Technology, 2022). This blockchain supports all existing Ethereum tools and enables faster and cheaper transactions. Known as an off-chain Layer 2 solution, its main objective is to solve scalability and usability issues without compromising decentralization and to leverage the existing developer community and ecosystem. Polygon is divided into three main layers: (i) Polygon SC; (ii) Heimdall, which is the PoS layer; and (iii) Bor, the layer that produces blocks.

**Klaytn** is designed for adoption by service providers and enterprises, so its architecture promises to provide the high level of flexibility, expandability, and modularity needed for horizontal growth (Klaytn, 2022). Furthermore, Klaytn uses Istanbul BFT as a consensus algorithm, as it seeks to combine the benefits of public blockchains with the performance of consensus algorithms based on Byzantine Fault Tolerant (BFT). This combination achieves high performance and efficiency, building a public blockchain that maintains robust security and transparency while offering enterprise-grade performance and reliability.

**Celo** blockchain aims to enable a new field of accessible financial solutions, including mobile users, so end-users only need a mobile number to integrate into the Celo ecosystem. This technology uses a new address-based decentralized identity layer to map phone numbers to wallet addresses. Celo takes a full-stack approach, where each layer of the stack is designed with the end-user in mind, taking into account other stakeholders (such as network node operators), comprising the following layers: (i) Celo Blockchain, (ii) Celo Core Contracts, and (iii) Applications. Part of the Celo blockchain code is derived from the Ethereum network implementation to maintain full EVM compatibility for smart contracts. However, this blockchain uses a BFT consensus mechanism (Proof-of-Stake), introducing the concept of Validator Groups as intermediaries between voters and validators. In addition, it has a different transaction and block format, new client synchronization protocols and security mechanisms, payment method, and gas price.

## 2.2 Hyperledger Fabric

Among the tools and frameworks that help develop blockchain systems based on the Linux Foundation’s Hyperledger Project, we can highlight Hyperledger Fabric (HLF), an open-source framework for developing enterprise-level solutions and distributed applications based on permissioned blockchain. Through a modular and versatile design, HLF can satisfy a variety of industry use cases (Gaur et al., 2018).

Hyperledger Fabric comprises a series of components and services. For example, the peers constitute the main functions of the HLF network, which are maintaining shared records,

<sup>1</sup> <https://eips.ethereum.org/>

<sup>2</sup> <https://eips.ethereum.org/EIPS/eip-20>

<sup>3</sup> <https://eips.ethereum.org/EIPS/eip-721>

executing the chaincode, accessing ledger data, endorsing transactions, and interacting with applications. There are three types of peers in the HLF: (i) *endorser peers*, which receive the request to validate the transaction and execute the *chaincode*; (ii) *anchor peers*, which receive messages and send them to other peers in the organization; and (iii) *orderer peers*, which create, order, and attach blocks to the ledger. In addition, the flow of transactions between network components is managed by the *ordering service*, by peers, which are responsible for receiving state updates in the form of blocks from the *ordering service*, and finally, by chaincodes, which handle the business logic of the network running on peers.

The persistence of the HLF ledger is formed by: (i) a world state, which is a database containing the current values of a set of states in the ledger, and (ii) a blockchain, which is the log of transactions that records all the changes that resulted in the world state, being collected in blocks that are attached to the blockchain. The world state can frequently change as states can be created, updated, and deleted. On the other hand, the blockchain, once written, cannot be modified since it is immutable. The network maintains multiple copies of a ledger, consistent with all other copies through a consensus process. Another relevant component is the Membership Services Provider (MSP), given that HLF is a private and permissions-based network. The MSP works for digital certificate management, user IDs, and authentication of all network participants. Only members with known identities can execute transactions.

### 2.3 InterPlanetary File System (IPFS)

The InterPlanetary File System (IPFS) is a distributed system that allows access to and storage of files, websites, applications, and data in general, powered by its community (Benet, 2014). In short, IPFS requests several computers on the network to share the data of a given page, making any user a data provider. Each piece of data using the IPFS protocol has a Content Identifier (CID), the hash result of the stored data. This value is specific to each piece of data, even if it is smaller than the content itself. In addition, IPFS uses the Directed Acyclic Graphs (DAG) data structure, a type of graph that does not allow cycles, and the edges have direction.

IPFS uses Merkle-DAG, which are DAGs with each node having a CID. As each node in the DAG has information indicating who its descendants are, the value of the CID is directly related to the graph's topology. Thus, two nodes with the same CID build the same graph. This issue becomes a crucial part of the IPFS operation, as it makes it possible to efficiently synchronize different graphs representing the structure of the files in the protocol. When creating a file representation, IPFS first divides it into several blocks.

This assumption allows other file parts to be authenticated faster from different sources. Another essential feature is that if two similar files are stored in the protocol, they can share parts of the Merkle-DAG. For example, if a software project is updated, only files that

have changed will have their addresses changed. This capability provides greater efficiency when transferring different versions of a large project.

## 3 Related Work

This section will cover works related to our study, including an overview of other research proposals for document registration in the literature.

Martiri et al. (2018) provided a system architecture for a degree management system, which includes a plagiarism tool and a statistics module. The flow starts by extracting unstructured information from documents in .pdf format, which are then converted into a structured format resulting in a compact table with the necessary fields from the documents. Finally, the table data is encrypted and stored on the blockchain.

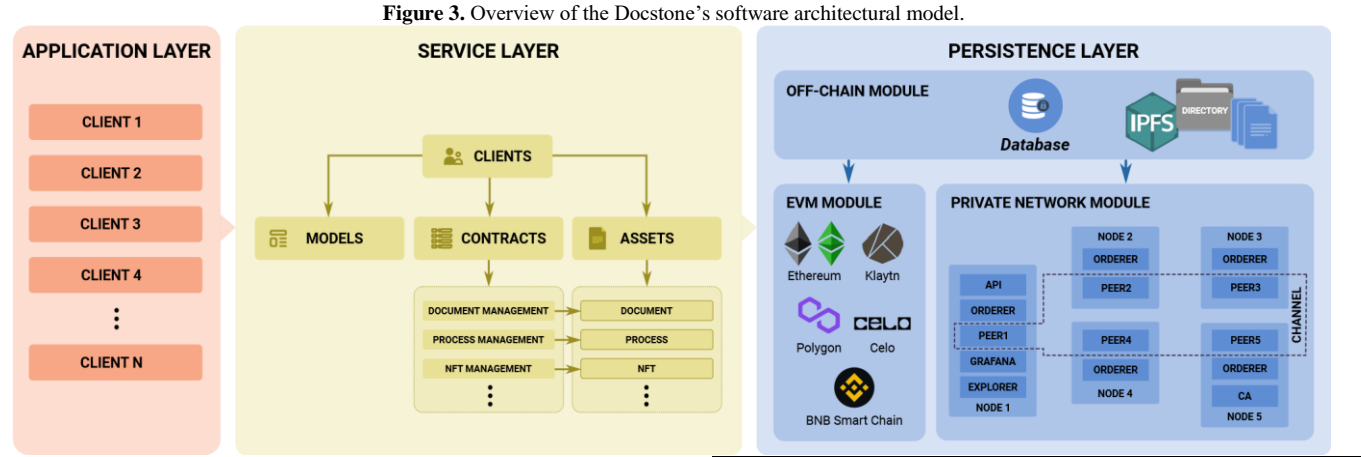
In turn, Bandara et al. (2020) presented Lekana, a document archiving platform using the Mystiko blockchain. The platform stores the file document information of an electronic invoice provider and their payloads (.pdf, .xml, image byte streams). The functions of creating, retrieving, and validating the hash chain of file documents are implemented with smart contracts. They also proposed a mechanism to build machine learning models.

Han et al. (2021) proposed a Decentralized Document Management System (DDMS) based on Ethereum to improve the security of digital documents. In summary, the DDMS works with a digital document encrypted employing a symmetric key, which is managed on a blockchain and rebuilt when the document is retrieved through smart contracts, which also perform access control.

More recently, Das et al. (2022) presented a solution based on Hyperledger Fabric with the main document management functionalities in AEC projects (Architecture, Engineering, and Construction), which involve information approval workflows, version management, and document lifecycle recording. Decisions in the document flow are recorded on the blockchain to verify the integrity of document versions and their log history.

As we can see, some of these related works stand out in their focus on safety issues. However, in contrast to the proposal of this work, they present non-customizable platforms when it comes to integrating different client applications. These approaches also did not present a feature to create generic models, for example. Finally, these works do not provide conditions to choose alternative blockchains or different assets quickly and in a personalized way.





## 4 Docstone: An Architectural Overview

This study proposes Docstone, an architecture based on blockchain that supports asset registration and validation services. The service includes asset (Document, Process, and NFT) management features, which can be easily integrated into different client platforms to create customizable asset templates that can be stored on one or more blockchains. As shown in Figure 3, the Docstone architecture is organized into three main layers: (i) Application Layer, (ii) Service Layer, and (iii) Persistence Layer. The last layer comprises three specific modules: Off-chain Module, EVM Module, and Private Network Module. We provide detailed information on this architecture in the following subsections.

### 4.1 Application Layer

In summary, the *Application Layer* comprises all client applications that connect to and use Docstone. In particular, this layer should provide a human-readable interface where users can perform and track their actions in their respective applications and businesses. The application layer does not implement a web interface, for example, of the Docstone itself, but instead abstracts those of the integrated clients.

We implemented Docstone to enable the main requirements of blockchain-based asset management to be generalized by different applications. To illustrate the generality of this service regarding the use cases that can be integrated, we can exemplify the types of client applications that have already connected to the proposed application at some level, such as (1) a system for copyright declaration certificates; (2) a system for registering certificates and diplomas of students in courses, schools, and universities; (3) a system for tracking protocols; and (4) a proposed system for registering news and images by press vehicles in the fight against fake news, among others.

Table 1 presents a subset of the main routes for integrating third-party interfaces into Docstone according to functionalities and SC type.

**Table 1.** Docstone’s routes and request for each functionality.

Functionalities	Method	Request	Description
Model	POST	/model	Creating models

Contract	POST	/contract	Deployment of a CI type in specific blockchains
	POST	/documents	Registration of a document
	GET	/documents	Search for a document
Document	POST	/documents/valid	Document validation
	POST	/processes	Registration of documents in a specific process
	GET	/processes	Search for documents from a specific process
Process	POST	/nfts	Minting a nft
	GET	/nfts	Search for a nft

The systems exemplified previously can be integrated with Docstone through available routes for each functionality, such as creating models, deploying contracts, registering and searching documents, processes, and NFTs, and validating documents. The requests in Table 1 are categorized based on their functionalities. For instance, `POST /contract` is used to deploy a contract into a blockchain, and it does not necessarily involve data storage. Similarly, `POST /nft` refers to the Process of “minting” an asset on the blockchain. In addition, we abstracted other functionalities in Table 1, such as functionalities related to managing Clients, Blockchains, and Activity Logs, among others.

### 4.2 Service Layer

The *Service Layer* refers to the implementation and execution of all the functionalities related to Document, Process, and NFT management. This capability is enabled in the API through four main classes: (i) Client; (ii) Model; (iii) Smart Contract (SC), and (iv) Assets. SC has specific types to manage Docstone’s assets (*i.e.*, Documents, Processes, and NFTs). Despite these SC and assets being pre-configured, we can further add other asset types to be managed by the API.

#### 4.2.1 Client class

The **Client** class represents each client application from the Application Layer, as exemplified in Section 4.1. The relationship model between the classes follows that:

- 1 Client has from 0 to  $N$  Models;
- 1 Client has from 0 to  $N$  SCs;
- 1 Client has from 0 to  $N$  Assets.
- 1 SC or 1 Asset has 0 to  $N$  Types.

#### 4.2.2 Model class

The **Model** class represents the structure of the assets that a Client will register on the blockchain through the API. In other words, a Model created by a Client consists of predefined specific attributes related to the asset and its general information, such as the Model's description. For example, if a Client registers student diplomas, they can create a Model that contains attributes such as course name, student ID, status, and grade, among others.

In addition to the attributes defined by the Client, the Model must contain the `idInternal` and `docHash` fields as mandatory fields. Docstone has these mandatory entries as a business rule because it uses off-chain mechanisms to map each Model attribute in the database with its respective content on the blockchain. In summary, off-chain mechanisms are related to outsourcing data storage outside the blockchain (Shukla and Samet, 2020). Hence, the `idInternal` is a document identifier related to the ID generated by the Client's storage systems, such as primary keys. The `docHash` refers to the hash of the Document or media generated by the Client application, usually in formats such as .pdf, .png, .jpeg, .mp3, among others. Thus, the Client must provide the `docHash` related to their documents or media as one of the inputs to the route for future validations. The Client can apply this `docHash` according to their design decisions, such as MD5 and SHA-256.

#### 4.2.3 Smart Contract class

The **Smart Contract (SC)** class refers to the implemented script code responsible for executing the main functions to manage each asset type in the blockchain. Initially, we implemented three SC types that can be selected according to what the Client wants to store. The SC types are (1) Document; (2) Process; and (3) NFT. Next, we detailed each implemented SC in the following.

**Document.** Each SC can be associated with at least one previously registered Model, *i.e.*, each SC stores Documents from one or more specific Model(s). The code presented in Listing 1 represents the SC for Document management designed to allow the generalization of document Models without the need to implement specific business rules. The SC is composed of two main functions: `insertDocument()` (line 19) and `searchDocument()` (line 26).

The Document SC has a struct (line 4), called '*Document*', which comprises a vector of attributes (`docsAttributes`) and CID as its main element. The `docsAttributes` stores the values and contents of the inserted documents. Each index of the vector corresponds to each attribute defined in the creation of the Model,

respectively, as illustrated in Figure 4. In turn, the CID is the hash provided by IPFS referring to the file associated with the respective registered Document.

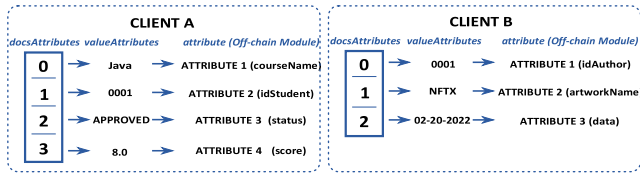
Listing 1: Docstone Document Smart Contract.

```
pragma solidity ^0.8.17;
contract Documents { address
public owner; struct Document {
string[] docsAttributes; string
cid;
}
mapping(string => Document) private
documents
; constructor() {
owner = msg.sender;
}
modifier onlyOwner() {
require( msg.sender ==
owner,
"Error: Only owner can call this
function"
);
_
}
function insertDocument( string
memory _idDocument, string[]
memory _docsAttributes, string
memory _cid
) public onlyOwner {
documents[_idDocument] = Document(
_docsAttributes, _cid);
}
function searchDocument(
string memory _idDocument
) public view returns (Document memory)
{ return documents[_idDocument]; }
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9 10 11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31

The use of *mapping*<sup>4</sup> ‘Documents’ (line 8) enables the Client to store  $N$  Documents from their respective identifiers. The `insertDocument()` function is responsible for registering data on the blockchain, which requests as input the Document ID and a vector of attributes related to the specific Document. In this case, it should be noted that only the owner of the address that deployed the SC can make calls to the SC functions. The `searchDocument()` function returns the inserted information of a Document from its ID.

**Figure 4.** Representation of the `docsAttributes` array of different Models.



**Process.** In this SC type, we start from the assumption that a Process contains different steps and, consequently, contains different documents. The API is implemented to deploy an SC when a client creates Process. In this sense, we codified an association between SC to a specific Process. Furthermore, we implemented the access control by the client application side for writing documents. Thus, SC 2 has the following functions: (i) `addAuthorizedAddress()`, (ii) `onlyAuthorized()`, (iii) `onlyOwner()`, (iv) `insertDocument()`, (v) `searchDocumentByStep()`, (vi) `searchDocument()`, and (vii) `getDocumentAmount()`.

We declared the `authorizedAddressmapping` (line 5), which the function will handle `addAuthorizedAddress()` in order to associate a client application address with an access status (true or false). Similarly to the first SC type, the Document's struct (line 7) consists of an array of attributes (`docsAttributes`) associated with the Document and its CID. In addition, each Document has the timestamp and the respective step that the Process was inserted.

**Listing 2:** Docstone Process Smart Contract.

```
pragma solidity ^0.8.17;

contract Processes { address
    public owner; mapping(address
    => bool) public
    authorizedAddresses;
    event DocumentInserted(string step, uint
    documentIndex);
    struct Document { string[]
    docsAttributes; string
    cid; uint time;
    string step;
    } mapping(string => Document[]) private
    steps; constructor() { owner = msg.sender;
    authorizedAddresses[owner] = true;
    }
    function addAuthorizedAddress(address
    _address) public onlyOwner {
    authorizedAddresses[_address] = true;
    }
    modifier onlyAuthorized() { require(
    authorizedAddresses[msg.sender],
    "Error: caller is not authorized"
    );
    _;
    }
    modifier onlyOwner() {
    require(msg.sender == owner,
    "Error:
    caller is not the owner");
    _;
    }
    function insertDocument( string
    memory _stepCode, string[]
    memory _docsAttributes, string
    memory _cid
    ) public onlyAuthorized returns (uint) {
    uint index = steps[_stepCode].length;

    steps[_stepCode].push(
    Document(_docsAttributes, _cid, block
    .timestamp, _stepCode)
    ); emit DocumentInserted(_stepCode,
    index); return index;
    }
    function searchDocumentsByStep(
    string memory _stepCode
    ) public view returns (Document[] memory) {
    return steps[_stepCode];
    }
    function searchDocument(
    string memory _stepCode,
    uint pos
    ) public view returns (Document memory) {
    return steps[_stepCode][pos];
    }
    function documentAmount(
    string memory _stepCode
    ) public view returns (uint) {
```

<sup>4</sup> Mappings are used to store data in key-value pairs.



9 10 11

12  
13  
14  
15  
16  
17  
18

19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29

30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40

41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57

```
return steps[_stepCode].length;
```

```
}
```

58  
59  
60  
61

The `insertDocument()` function (line 32) is similar to the one performed in Document Type. Still, the client application will be able to add to which step of the Process the inserted Document belongs. We point out that only the addresses registered in `authorizedAddress` can write a Document.

There are two types of reading, `searchDocument()` (line

45) and `searchDocumentsByStep()` (line 50). The first performs the search for information on a specific document based on the step and position in which the Document is. Based on this information, the company passes it transparently to the developer. A cross-referencing between the lead's database and blockchain is carried out, and the searched Document is returned. The second one returns data from the documents of a specific step. Finally, the `getDocumentAmount()` (line 56) is an auxiliary function that returns the number of steps in a process.

**NFT.** As presented in the SC depicted in Listing 3, this type is related to the insertion of documents (or other types of assets) and its conversion to the ERC-721 standard. In other words, the asset stored through this SC type will be transformed into a tokenized version. To implement this SC, we used the OpenZeppelin library (imported in lines 2 and 3), which has a collection of code to implement some standards, including the ERC-721.

Listing 3: Docstone NFT Smart Contract.

```
pragma solidity ^0.8.17;
import "@openzeppelin/contracts/token/ERC721/ERC721.sol"; import
"@openzeppelin/contracts/utils/Counters.
sol";
contract Nfts is ERC721 { using
Counters for Counters.Counter;
address public owner;
Counters.Counter private _tokenIds;
constructor() ERC721("MyNFTs", "NFT") {
owner = msg.sender;
}
struct NFT { string
cid; string[]
metadata;
} mapping(string => NFT) private
_tokenData; function mint( address to,
string memory tokenId, string memory
tokenCID, string[] memory metadata
) public returns (uint256)
{ require( msg.sender
== owner,
"Error: only the contract owner can
execute this function"
);
require(bytes(tokenId).length > 0,
"Token
ID cannot be empty"); require(
bytes(_tokenData[tokenId].cid).length
== 0,
"Token ID already exists"
);
_tokenIds.increment(); uint256
newItemId = _tokenIds.current();
_mint(to, newItemId);
```

1  
2  
3  
4  
5  
6  
7  
8  
9

```

10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
    _setTokenData(tokenId, tokenCID,
        metadata );

    return newItemId;
}
function _setTokenData(
    string memory tokenId,
    string memory cid,
    string[] memory metadata
) private {
    _tokenData[tokenId] = NFT(cid,
        metadata);
}
function getTokenData(
    string memory tokenId
) public view returns (NFT memory)
{ return _tokenData[tokenId]; }
}
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50

```

As we can see, the “NFTs” SC inherits the ERC-721 standard as NFTs usually follow this standard. Its constructor (line 8) initially takes a `name` and a `ticket` for the tokens initially named with “MyNFTs” and “NFT”. Then, the struct of the NFT (line 11) consists of its `CID` and `metadata` with other information. Finally, the `mint()`

function (line 16) creates the token and sends it to a specified address. In addition, some checks are performed to the creation of the NFT: (i) if the client application running the SC is the owner, (ii) if the token ID is greater than 0, and (iii) if the token ID exists.

Finally, the `_setTokenData()` (line 38) is an auxiliary function to store data related to the minted token. In turn, the `getTokenData()` (line 45) function returns information about a specific token.

Unlike the previous study, where there was no possibility to choose a smart contract, these adaptations allow that other smart contracts and asset types to be also integrated into Docstone in a customizable way.

#### 4.2.4 Assets class

Finally, the **Asset** class consists of an asset’s current content/value (Document, Process, or NFT). Each asset information is managed according to the Model previously created.

### 4.3 Persistence Layer

The *Persistence Layer* has three modules to support storing document content: (i) Off-chain module; (ii) EVM module; and (iii) Private Network module.

#### 4.3.1 Off-chain module

We are using two types of persistence in this module (Relational Database and InterPlanetary File System), as detailed below:

**Relational Database.** We use an auxiliary database to register and manage application Clients and store their credentials, API access tokens, and other relevant information (such as sensitive data). Thus, the database also stores private information and Model attributes created to perform data crossreferencing between the database and the blockchain. With this off-chain approach, there is no need to insert raw records into the blockchain since it stores only its main pointers and hashes.

**InterPlanetary File System (IPFS).** We also decided to use a peer-to-peer distributed file system such as IPFS to store media files (`.txt`, `.docx`, `.pdf`, `.mp4`, among others), given the limitations of file types for storage on the blockchain (which preferably stores metadata in text format). Like the blockchain approach, each user on IPFS can instantiate a node. In our case, the server running Docstone also runs a parallel process with an IPFS node. This node is explored to maintain a local copy distributed to the IPFS network. The *ipfs-core* library was used to carry out this integration, which is the implementation of the IPFS Core API written in JavaScript without depending on other languages/implementations. This library provides all the necessary functions to integrate IPFS into an application.

Regarding the implementation model for Docstone and IPFS integration, we decided to use the concept of IPFS directories. We developed a mechanism for automatically creating IPFS directories mapped to each smart contract. In addition to storing independent documents, Docstone creates directories to organize documents, processes (grouped documents), and NFTs collections. For example, when deploying an SC referring to Process A, a new directory is created to store all the respective documents related to this Process. Figure 5 presents an example of a directory on IPFS regarding a process with its documents.

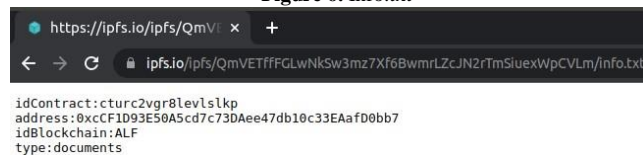
Figure 5. Directory created with the respective documents.



Index of /ipfs/QmVETffFGLwNkSw3mz7Xf6BwmrLZcJN2rTmSiuxWpCVLm 20 MB		
QmVETffFGLwNkSw3mz7Xf6BwmrLZcJN2rTmSiuxWpCVLm		
dcure2v1xolevwekf9	QmUg...TeF3	266 kB
dcure2v5gclevm4o6x.jpg	QmUu...E5gE	39 kB
dcure2v5iclevsvuv3	QmUu...E5gE	39 kB
dcure2v5iclevswlns	QmUg...TeF3	266 kB
dcure2vcw4lew7cy3j	QmZF...kgcy	6.8 MB
dcure2vgq4levyx0lw	Qmbz...ZYPd	6.1 MB
dcure2vgq4lew5f1gt	QmZF...kgcy	6.8 MB
dcure2vgr8levlvi74.jpg	QmUu...E5gE	39 kB
info.txt	QmaT...C4Te	121 B

As shown in Figure 5, each directory has a CID (e.g., /ipfs/QmVE...CVLm), just like each Document (e.g., QmUg... TeF3). By default, files are named with the `idDocument` generated by Docstone to simplify automation (e.g., `dcure2vaxolevwekf9`). However, this naming convention can be more readable with other document titles. Therefore, when Docstone creates a directory, it also generates an initial file (`info.txt`) with general information, such as the associated smart contract's ID (`idContract`) and its address in the blockchain where it was deployed (`idBlockchain`), as well as the SC type (Documents, Processes, or NFTs).

Figure 6. Info.txt



```
idContract:cturc2vgr8levlslkp
address:0xcCF1D93E50A5cd7c73DAee47db10c33EAaf08bb7
idBlockchain:ALF
type:documents
```

#### 4.3.2 EVM module

The non-sensitive information defined from the Model and stored in the asset is inserted into the blockchain. Among the design decisions of this solution, we point out that the service allows the insertion of information into one or more blockchains to enable a client application to be integrated into

Docstone using a blockchain that best suits their demands. The option of choosing blockchain (s) can be oriented by different criteria, such as:

1. Transaction speed;
2. Transaction cost;
3. Security;
4. Scalability;
5. Consensus algorithm;
6. Technologies used by each blockchain;
7. Network's reputation in the community and engagement;
8. Level of decentralization;
9. Number of transactions per second;
10. Application business context, and others.

Considering that the SC code is written in a stack-based bytecode language and executed in the Ethereum Virtual Machine (EVM) through the Solidity language (Tikhomirov, 2017), it can be executed by all public blockchains compatible with EVM integrated into this solution, for example, Ethereum and Ethereum Classic; BNB Smart Chain; Polygon; Klaytn; and Celo.

#### 4.3.3 Private Network module

A single organization-based Hyperledger Fabric Private Network was also implemented to integrate the blockchains connected to the API. We chose this due to support for confidential transactions, given the ability to configure access to specific transactions for predefined users. Additionally, one can highlight that cryptocurrencies are unnecessary for performing operations. Figure 3 presents the Private Network's structure which has five nodes, each containing at least one peer and one orderer.

The network's peers are only of the anchor type since it is a single organization, in which there is no need to transmit messages to peers from another organization. Additionally, the network uses Transport Layer Security (TLS), which requires configuring a TLS CA and its use to generate TLS certificates. Finally, explorer and Grafana have also been implemented to monitor the network hosted on one of the five nodes.

## 5 Docstone Demonstration

Figure 7 illustrates the main flow through a client application's registration of information on the blockchain using Docstone. To exemplify this use case, we adopt the process of data registration related to an artwork copyright certificate system. Indeed, our API does not restrict other types of applications since it is possible for all those with a similar flow to be integrated into Docstone.

Initially, the Client was registered with access credentials to submit requests to Docstone. In **Step 1**, the Client creates an artwork certificate model. The model structure consists of the following attributes: `idAuthor`, `idArtwork`, `description`, and `artworkType`. In addition, the Client must pass the `idInternal` referring to the

identifier in the application's database and the `hashDoc`, which consists of the hash of the certificate media to be inserted. When the client request `POST \models`, Docstone creates a new model with the informed data and receives a `modelCode` as an identifier. Optionally, the `attrNotNull` field establishes the required attributes among those defined in the model. Considering the proposed example, `idAuthor` and `idArtwork` are mandatory attributes.

In **Step 2**, the Client requests the `POST \contracts` to deploy the SC onto a specified blockchain indicating two parameters: (i) what blockchain the SC will be deployed and, consequently, where the artwork certificate information will be stored and, (ii) the SC type (code) that will be used. In this case, Docstone addresses the available blockchains listed in Section 4.3.2. We will use the Ethereum (ETH) network for the current demonstration. In addition, we also choose "documents" from the available types (Documents, Processes, and NFTs) when deploying the SC, as described in Section 4.2.3. Finally, Docstone returns a `idContract` after being deployed in the blockchain.

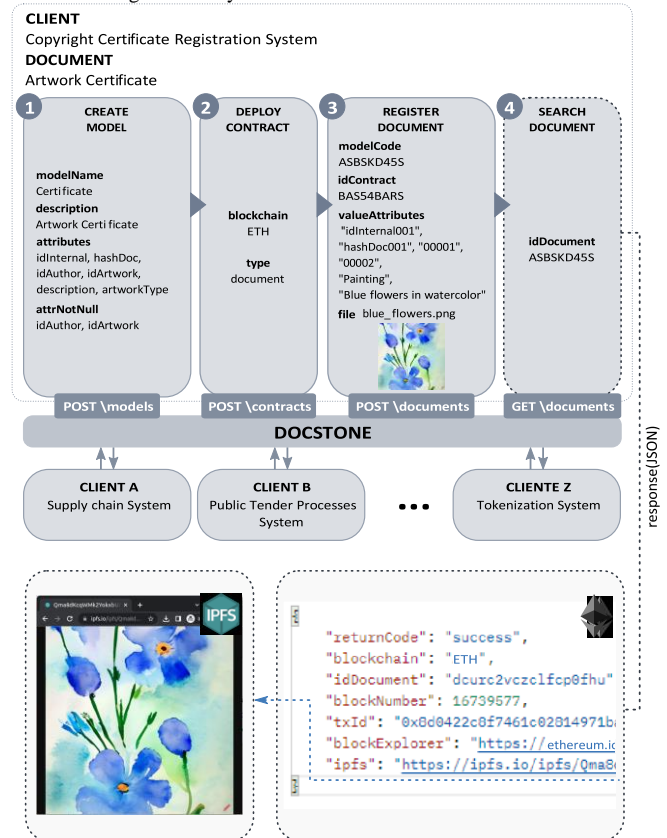
In **Step 3**, the API receives the `idContract` and `modelCode` to identify the SC already deployed and the type of model that represents the format in which the information will be stored when requesting the `POST \documents`. At this point, the document's content will be inserted into the blockchain. For example, the stored artwork certificate information indicates that an author with `idAutor` '0001' created an artwork consisting of `idArtwork` '0003', whose `typeWork` is a 'Painting' and the description is 'Blue flowers in watercolor'. The certificate's information is the metadata related to a specific document. Additionally, the Client can submit the media or file representing the document (or artwork, in this example) in the file field<sup>5</sup>. In other words, the client application can add the metadata (stored in the blockchain) and the digital 'Painting' file (stored in the IPFS), as shown in Figure 7.

Finally, this record can be queried through the `GET \documents` request (**Step 4**) where Docstone returns the metadata from blockchain and the file of the artwork from the IPFS. In addition, several other requests can be made to handle these artwork certificates, such as the validation request. In this case, the client application must inform the document `hashDoc` of the media. In turn, Docstone returns whether the file hash is valid compared to the previously registered on the blockchain. This requirement also allows verification of the file by an external user in cases in which it is necessary to consult or validate some information or documents.

In addition to the discussed functionalities, the service has other requirements and enables registering data in one or more blockchains. Hence, Docstone allows the deployment of other SC types, which may require some changes and adaptations in the flow. For example, the client application can deploy a processes-type SC instead of a documents-type SC (with `/POST contracts`). In this case, when adding each document, it is necessary to inform the `idProcess` and the specific step to which the inserted document belongs. In this sense, we can have a set of documents associated with a single process, different from the documents-type SC, where the documents are independent. For instance, these cases can be used for government procedures, such as 'Public Tender Processes', in which a process is usually opened and consists of creating different stages and inserting documents until completion.

On the other hand, using the SC of NFTs enables tokenizing documents (or other types of assets) in the standard already widely used among Web3 and blockchain users, the ERC-721, to transfer ownership and commercialization of the tokenized object. However, the commercialization of the document tokenized is different from the focus of this research. Furthermore, other SC types (new encodings), such as recurrent usage standards, can be added to the blockchain with the proper adaptations.

**Figure 7.** Demonstration of Docstone integrated with a Copyright Certificate Registration System.



<sup>5</sup> The picture presented in Figure 7 was generated in <https://www.crayon.com> (an Artificial Intelligence image generator) prompted with the text "Blue flowers in watercolor".



## 6 Empirical Study

We conducted a computational experiment using a set of analyses and a performance metric. This work developed an API as Proof-of-Concept (PoC) for experimentation with the technologies described in Table 2.

**Table 2.** Tools used in the implementation of the PoC.

Tools	Brief description
<b>Solidity and Go</b>	Languages used to implement smart contracts based on EVM and HLF, respectively.
<b>Web3.js and Caver.js</b>	Collection of libraries that allow interaction with a local or remote Ethereum node. Caver.js is used on the Klaytn blockchain.
<b>Sepolia Alfajores</b>	Test networks of the used blockchains. Normally is used by developers to run tests without using real cryptocurrency.
<b>Node.js and Express.js</b>	Node.js is a server-side Javascript execution environment. Express.js is a web framework based on the core Node.js HTTP module and middlewares.
<b>Docker</b>	A set of virtualization products for delivering software in packages called containers, which can be orchestrated by the Docker Swarm tool.

### 6.1 Experiment Setup

We conducted this experiment to analyze the performance quality attribute considering the Testnets of two public blockchains: Sepolia (Ethereum 2.0) and Alfajores (Celo). It is worth noting that our previous studies Soares et al. (2022b) did not include testing on Sepolia and Alfajores. Sepolia is a testnet for a new version of Ethereum that was launched on December 1, 2020, and utilizes the PoS consensus algorithm. Meanwhile, Alfajores is a Celo testnet that focuses on sustainable and democratic solutions, offering a new dimension to our analysis of blockchains.

In the version of this article, we only use these two blockchains (of six integrated into the Docstone), given that our main goal here is to analyze the difference between SC types (Documents, Processes, and NFTs). Another difference is that we do not execute the single organization network presented in the previous study as we already compared the performance using public and private blockchains in (Soares et al., 2022b). We overview our four new primary analyses in Table 3.

After delimiting what blockchains will be investigated, we established three independent variables for this experiment: “*number of requests*”, “*type of request*”, and “*blockchain*” and “*latency*” metric as a dependent variable. We adopted JMeter 5.020 (Nevedrov, 2006) to simulate loads and send them to Docstone. In order to represent an asset model, we followed the attributes defined below:

Listing 4: Model created for performance testing.

```
{
  "description": "Standard model for
testing.",
  "attributes": "idInternal;hashDoc;field1",
  "modelName": "standard01",
  "attrNotNull": "field1"
}
```

1  
2  
3  
4  
5  
6

**Table 3.** Overview of the conducted analyses.

Analysis	Requests	Blockchain
1) POST requests (write) analysis regarding the use of IPFS	POST /documents (without IPFS)	Sepolia Alfajores
	POST /documents (with IPFS)	Sepolia Alfajores
2) POST requests (deployment) analysis of smart contract types	POST /contracts (Documents)	Sepolia Alfajores
	POST /contracts (Processes)	Sepolia Alfajores
	POST /contracts (NFTs)	Sepolia Alfajores
3) POST requests (write) analysis of smart contract types	POST /documents (with IPFS)	Sepolia Alfajores
	POST /processes (with IPFS)	Sepolia Alfajores
	POST /nfts (with IPFS)	Sepolia Alfajores
	GET /documents	Sepolia Alfajores
4) GET requests (read) analysis of smart contract types	GET /processes	Sepolia Alfajores
	GET /nfts	Sepolia Alfajores
	GET /nfts	Sepolia Alfajores

The defined scenario considers a specific client application that makes requests every 5 seconds in a serialized way. We chose this approach as we aim to evaluate only the behavior of Docstone being used by a company. For analysis purposes, we executed 100 requests made by this client application. The experiments were conducted on a virtual machine with the *VM.Standard.E4.Flex* from Oracle Cloud with 8GB of RAM, 2vCPUs, and a broadband internet connection of 1 Gbps.

Wilcoxon (WC) statistical tests were applied to the results of each blockchain and SC Type to identify the occurrence of statistical differences between the samples, considering a confidence level of 95% using the Bonferroni correction. We also used Vargha-Delaney as the effect-size to return the relative number of times one case type produced higher values than the other. Additionally, we generated a boxplot for each analysis to visualize and analyze the distribution of the set of results.

### 6.2 Performance Evaluation

In line with Table 3, we present below our results grouped by four different analyses.

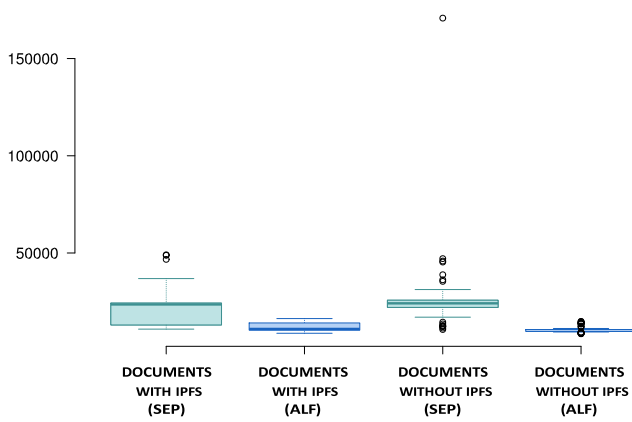


### 6.2.1 POST requests (write) analysis regarding the use of IPFS

Figure 8 shows the difference between registering with and without IPFS to store media files referring to documents, one of the main changes related to previous work (Soares et al., 2022b). Thus, we want to assess whether the use of IPFS causes any change in terms of latency time. In this case, we evaluate only for writes of the Document asset type.

The present study reports on the latency performance of `POST /documents` requests with and without IPFS. In Sepolia, the results demonstrate that the median latencies for requests with IPFS and without IPFS were 23531.50ms and 24104ms, respectively. While the difference between these

**Figure 8.** Latency of the `POST \documents` request on the Sepolia and Alfajores blockchains with and without IPFS.



medians may appear relatively small, the use of IPFS was associated with greater variability in the latencies of requests in Sepolia. This finding suggests that, when utilizing the `POST /documents` route in Sepolia to store a media file, there may be a higher variance in latency times when employing IPFS instead of simply sending the file's metadata without requesting the IPFS.

The statistical analysis presented in Table 4 supports these results, which indicate a statistically significant difference in the use of IPFS in Sepolia, with a significance level of 95%. However, the analysis also demonstrates that IPFS resulted in higher latency times only 36% of the time and that, in general, IPFS did not significantly increase latency times in Sepolia.

**Table 4.** Latency of the `POST \documents` request on the Sepolia and Alfajores blockchains with and without IPFS.

	Without IPFS			
	Alfajores		Sepolia	
	WC	Â	WC	Â
With IPFS	6.8e-08	0.72	0.0013	0.36

Regarding the results of the Alfajores, the median latencies for `POST /documents` requests with and without the use of IPFS were 10941ms and 10270ms, respectively. With this blockchain, we noticed a more significant variability in latency results compared to not

using IPFS, although with a smaller amplitude than in Sepolia. As for the results of the statistical tests carried out with Alfajores, we observed a statistically significant difference (see Table 4), considering a significance level of up to 99%. Approximately 72% of the time, using IPFS resulted in higher latency times than when IPFS was not used at Alfajores. We can see that using IPFS for media storage negatively impacted the latency time for the Alfajores network.

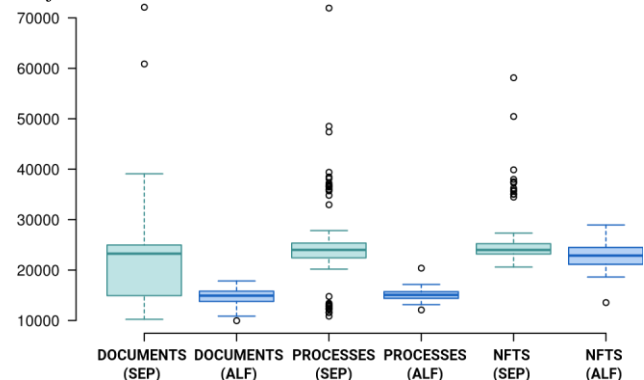
Regarding network performance, Alfajores exhibits a set of latencies that are noticeably distinct from those observed in Sepolia, despite having lower values. Statistically speaking, Alfajores only outperforms Sepolia using IPFS approximately 12% of the time, and without IPFS, that figure drops to a mere 2%.

### 6.2.2 POST requests (deployment) analysis of smart contract types

In Figure 9, the different SC types in the Sepolia blockchain presented similar median latency times during the deployment, with values of 23256.50ms, 23993.50ms, and 23988.50ms, respectively, to Documents, Processes, and NFTs. However, we noticed that the SC Document presented a more significant variability of latency values when compared to other SCs. Some responses took much longer latency than others (and vice versa). This behavior is particularly relevant, given that the Document and Process SC implementation have similar logic, with little change between functions.

Moreover, we discovered a statistically small difference between NFTs and Documents (for a confidence level of 95%) by analyzing the statistical tests in Table 5. In the other comparisons, there is no statistically significant difference.

**Figure 9.** Latency of the `POST \contracts` request on the Sepolia and Alfajores blockchains.



Regarding the latency values of the request at Alfajores, we note that only the Documents and Processes contracts have similar median values (14932ms and 15074.50ms). In contrast, the median latency value of the NFT deployment is almost 1.5 times higher. Therefore, we realized that the result confirms the conclusion obtained in the statistical test presented in Table 5, where it is possible to verify that there is no statistical difference between Documents and Processes (value in red). In addition, the other statistical results showed

that the NFT contract had the worst latency time compared to the others, approximately 99% greater than the latency time of the Documents and Processes contracts.

**Table 5.** Latency of the `POST \contracts` request on the Sepolia and Alfajores blockchains.

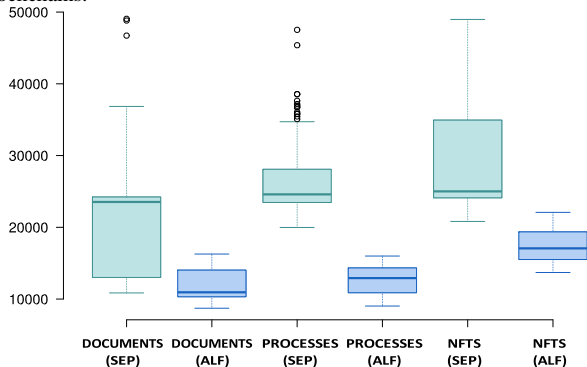
	Alfajores				Sepolia			
	Documents		NFTs		Documents		NFTs	
	WC	Â	WC	Â	WC	Â	WC	Â
<b>NFTs</b>	2,00e-16	0,9921	-	-	0.004	0.62	-	-
<b>Processes</b>	<b>0.98</b>	<b>0.54</b>	2,00e-16	0,01	<b>0.27</b>	<b>0.56</b>	<b>0.61</b>	<b>0.44</b>

Regarding the analysis between the networks, the Alfajores network stands out with better performance. For the three types of assets (Documents, Processes, and NFTs), there is a statistical difference using Sepolia since it has lower latency results in most cases. In turn, Sepolia spends more time to complete the deployment, approximately 77%, 85%, and 67% of the time for each asset type, respectively.

### 6.2.3 POST requests (write) analysis of smart contract types

According to Figure 10, the different assets in the Sepolia network present approximate median values for Document, Process, and NFT, 23531.50ms, 24590.00ms, and 25006ms, respectively. However, we found significant statistical differences between these assets. For example, we verified that 79% of the time, between Document and NFT, the time to write information through the POST route was more significant for NFT. Likewise, we obtained a statistical difference contrasting Processes and Documents, where Processes had higher results 72% of the time than Documents. However, no significant difference was found between Process and NFTs in the Sepolia network, according to the values in red in Table 6. These results demonstrate that the latency time between these assets is similar when using Sepolia.

**Figure 10.** Latency of the write requests on the Sepolia and Alfajores blockchains.



On the Alfajores network, 50% of the latency values are concentrated below 10941.00ms, 12919.50ms, and 17064.50ms for Documents, Processes, and NFTs,

respectively. These values are well below what was obtained with the Sepolia network. We also noticed a variability of latency values between the Documents and Processes assets similar to the Sepolia network. When we compare the performance of assets on the Alfajores network, both Processes and Documents again stand out concerning the NFT asset. However, the latter has higher latency times of more than 90% of the time compared to the other two assets, with approximately similar times and statistically smaller than the NFTs.

However, in the Alfajores network, we obtained a statistically significant difference between Processes and NFTs, indicating that the latency time of the Processes asset was

higher than that of the NFT asset only 4% of the time. Therefore, we conclude that the Documents asset was the fastest concerning the latency time in writing information as in the Sepolia network. However, the Processes asset has a shorter time than the NFTs in the Alfajores network.

Finally, regarding the analysis between the networks, the Alfajores network stands out again with the best performance in the three types of assets, with Sepolia being greater in 88%, 100%, and 99% of the times for each type of asset, respectively.

**Table 6.** Latency of the write requests on the Sepolia and Alfajores blockchains.

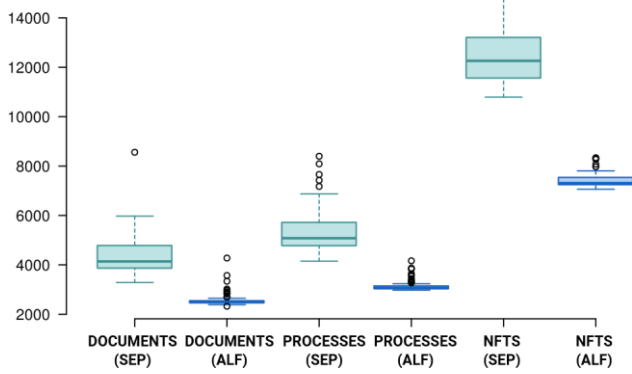
	Alfajores				Sepolia			
	Documents		NFTs		Documents		NFTs	
	WC	Â	WC	Â	WC	Â	WC	Â
<b>NFTs</b>	2.00e-16	0.95	-	-	8,00e-13	0.79	-	-
<b>Processes</b>	0.002	0,61	2.00E-16	0.04	8.50e-08	0.72	<b>0,12</b>	<b>0,41</b>

### 6.2.4 GET requests (read) analysis of smart contract types

Figure 11 presents an analysis of latency performance for different asset types and blockchains, as demonstrated in Figure 11. Our findings reveal that each box in the chart represents a statistically significant difference related to asset types and blockchains. Furthermore, our examination of the data presented in Table 7 permits us to make robust statistical conclusions regarding the performance of the various asset types. Specifically, our analysis indicates that Documents exhibit superior performance compared to other asset types, with significantly lower latency values. In contrast, NFTs are associated with latency values approximately 100% higher than Documents on both networks. While exhibiting higher latency values than Documents, processes demonstrate strong performance, with latency values around 82% on Sepolia and 97% on Alfajores relative to Documents. Notably, the Processes asset type also outperforms NFTs regarding search time, requiring only 100% of the time on both networks.

Our results suggest that NFTs are the asset type most associated with longer search times and higher latency values. At the same time, Documents and Processes demonstrate superior performance in this regard.

**Figure 11.** Latency of the read requests on the Sepolia and Alfajores blockchains.



**Table 7.** Latency of the read requests on the Sepolia and Alfajores blockchains.

	Alfajores				Sepolia			
	Documents		NFTs		Documents		NFTs	
	WC	Â	WC	Â	WC	Â	WC	Â
<b>NFTs</b>	2,00e-16	1	-	-	2,00e-16	1	-	-
<b>Processes</b>	2,00e-16	0,9706	2,00e-16	0	4.4e-15	0,82	2,00e-16	0

Similar to the previous analyses, we notice that the results of the assets with Sepolia present latency values with more significant variability than those coming from Alfajores. For example, while the standard deviation for the set of latencies of NFTs in Sepolia is approximately 1072ms, in Alfajores, it is 272ms, almost 4x more minor than in the first blockchain.

Lastly, on the analysis between the networks, as well as the deploy and write requests, in the read requests, the Alfajores network stands out with the best performance in the three assets. In the latter case, Sepolia is more significant by 99%, 99%, and 100%, respectively.

### 6.3 Discussion and Lessons Learned

We briefly discuss in this section the highlights of Docstone architecture and the lessons learned. To this end, we also analyze our proposal based on Wöhrer et al. (2021), which proposed a framework for decisions, issues, options, and conceptual components concerning the design of blockchain-based architectures and applications.

#### 6.3.1 Performance

Specifically, public blockchains usually have more challenging scalability than private blockchains. Hence, the performance of networks can vary according to several factors. These causes do not refer only to the applied consensus algorithms but also to other blockchain scalability limits that may be related to the size of the data in the blockchain, the processing rate, and the latency of the

data transmission data (Koteska et al., 2017; Xie et al., 2019).

In addition, we recognize that test networks can differ greatly from mainnets in terms of performance. For example, while Ethereum has 6049 nodes<sup>6</sup> in its structure, Sepolia (Ethereum’s testnet) has 1972 active nodes<sup>7</sup>. When writing this study, we collected these numbers, which are subject to change. Certainly, running the experiments on mainnets would also provide relevant results more adherent to reality.

The variability in latency values is another important point depicted by our results. In most of the analyses (Sections 6.2.2, 6.2.1, 6.2.3, and 6.2.4), the Sepolia blockchain shows a higher range of values, resulting in less stability in terms of request latency time, which is common in blockchain networks. On the other hand, we also observe that Alfajores demonstrated greater stability in response flow, maintaining consistent values, especially for write and read requests. In practice, when developers and researchers in this study requested operations on blockchains, the results for Sepolia were not consistent related to the latency times. Sometimes, the response returns within a few seconds, while in other situations, we may need to wait for a significant amount of time. In contrast, on Alfajores, the responses were consistently returned within approximate latency times.

#### 6.3.2 Decentralization level

The level of decentralization in the Docstone architecture was a careful design decision, with the option of either complete or partial decentralization (Wöhrer et al., 2021). In this sense, Docstone adopts a semi-decentralized or hybrid architecture with centralized components to support certain functions, such as storing sensitive information (as described in Section 4). This decision was made to balance the advantages of decentralization with the technical limitations and usability challenges that still exist today. While partially decentralized architectures reduce the need for trust in fully decentralized applications, centralized components can still provide practical benefits in certain scenarios (Wöhrer et al., 2021).

#### 6.3.3 Identity Provisioning, Key Management, and Transaction Handling

Docstone does not yet implement the mapping of users through their specific addresses (address in Solidity) in blockchain through their wallets. In other words, handling the blockchain identity for the user is the responsibility of Docstone itself through managing data in an off-chain approach, without handling the custody of the users’ secret keys, for example. Indeed, Docstone manages its key pairs, signing and forwarding information (transactions) to the blockchain. Thus, implementing key management by users is

<sup>6</sup> <https://ethernodes.org/>

<sup>7</sup> <https://sepolia.beaconcha.in/>

a strong lead for Docstone to move towards a fully decentralized version.

Furthermore, Docstone does not use its wallets to map users via their specific blockchain addresses (address in Solidity). This decision means that the handling of a user's blockchain identity is the responsibility of Docstone itself through off-chain data management, without handling the custody of users' secret keys, for example. However, Docstone manages its key pairs and signs and forwards information (transactions) to the blockchain. We acknowledge that implementing key management by the users is a crucial step for Docstone toward achieving a fully decentralized version.

Regarding transaction handling, our development process also faced challenges similar to those described by Wöhrer et al. (2021), such as nonce errors, network congestion, peer loss, and transaction loss due to sudden price increases. Transaction Managers help to overcome these issues by controlling how transactions are signed and transmitted to the blockchain network. However, these aspects are still under development and generally come with additional costs. For PoC purposes, adopting specific Transaction Managers (EthVigil, for example) is quite challenging. However, other techniques that follow these principles were applied, such as estimating adequate transaction costs to ensure sufficient funds for execution and managing nonce and keys.

#### 6.3.4 Connection to blockchain

In particular, the interaction with the blockchain was carried out through blockchain endpoints, which are devices or nodes running software that implements the blockchain protocol. To achieve this capability, Docstone explores providers (such as Infura) that enable users to interact with the blockchain without setting up their nodes. We clarified that the creation of nodes for the public blockchains integrated into the API was not initially necessary, as our goal was not to implement a truly private, self-sufficient, and trustless network for the use of public blockchains (which is also possible), but, in fact, to the use of the HLF network with permission control properties.

#### 6.3.5 Application Logic and Storage

We employed on-chain (through SCs) and off-chain (in the back-end, database, and other storage systems) strategies for logic implementation and storage. As previously explained, off-chain techniques were used for user control and authentication to the API in the application logic. In regard to storage, the goal was to preserve sensitive customer information and attempt to outsource the storage location of raw data, such as large amounts of data and media files.

With respect to back-end implementation, the code was optimized by using *Web3.js*. We also created middleware as well as unique and modularized flows since most

integrated blockchains follow the EVM standard. Consequently, a specific blockchain can be defined in function inputs and outputs generated similarly for all blockchains. We pointed out that, in order to improve security requirements, data stored in the database was encrypted, and access tokens were generated for customer use to routes. Finally, special tools were approached for API development, where the proper definitions of resources, collections, endpoints/URLs, use of HTTP status codes, and other important components in the development of RESTful APIs were established.

#### 6.3.6 Minimization of redundancies and the use of blockchain's libraries

Most implemented blockchains can use the web3.js library as it follows the Ethereum standard (EVM-based). Therefore, as a best practice, we architected the code project to avoid redundancies related to using blockchain libraries. Instead of each blockchain using the Web3.js library separately, we grouped the blockchains by library type and enabled middleware to perform optimized coding. For instance, there are currently three groups of blockchains: 1) *web3*, a group with unique functions, code, and web3.js libraries to connect to Ethereum, Polygon, BNB Smart Chain, or Celo (and its testnets), 2) *caverjs*, a group with unique functions, code, and caver.js libraries to connect to Klaytn, and 3) *hlf*, with specific coding to integrate with HLF.

Suppose we decide to integrate the Huobi Eco Chain (HECO)<sup>8</sup> in the future, for example. In that case, we can easily add it to the existing *web3* group without making any changes to the code, as we only need to provide the blockchain's specific configuration parameters. Some of the parameters to add a new blockchain into Docstone are *name*, *chainId*, and *RPC endpoint*, for example. However, if we want to include Solana<sup>9</sup> in Docstone, we must implement a new group that includes the specific libraries. Once developed, these libraries can be used by any other blockchain based on Solana, with minimal or no code modifications necessary.

#### 6.3.7 Using different assets and SCs

Unlike previous studies (Soares et al., 2022a,b), the current research employs two additional types of assets and smart contracts related to Processes and NFTs. Regarding their deployment in Sepolia, there is a similarity between the asset types, with the document SC latency time showing inconsistency in values. At Alfajores, the NFT's SC deployment latency time was considerably higher than that of Document and Processes. An interesting observation is that the NFT SC imports other libraries with other legacy contracts, which could lead to longer latency times for both

<sup>8</sup> <https://www.hecochain.com/>

<sup>9</sup> <https://solana.com/>

blockchains. However, we observed this behavior only in Alfajores.

Regarding writing and reading different assets, the NFTs had higher latency times in Sepolia and Alfajores. Nevertheless, Documents had the lowest latency time in both networks in writing and reading requests. These findings are consistent with the complexity of SCs implementation, as Documents is the simplest SC to implement and involves fewer operations when compared to Process and NFTs. However, we emphasize that the complexity of SCs affects the cost of blockchain fees.

In short, the difference in SC implementation impacts more than the types of assets themselves. Depending on the type of asset, the logic implemented in the SC may require a series of business rules that demand greater complexity. The impacts from the asset can be perceived depending on the content inserted, as the loads can be small or large.

### 6.3.8 Distributed File Storage System

Different approaches have been proposed in the literature to be used as auxiliary storage (as off-chain mechanisms) since storing media files is still a limitation in the blockchain. Eberhardt and Tai (2017) coined the term Content-Addressable Storage Pattern to the pattern that uses distributed storage systems to outsource the storage location of raw data. In this sense, we use IPFS in our solution to use auxiliary storage for media files. However, we faced some challenges during the initial integration. In contrast with blockchains, IPFS is not immutable whether the file does not maintain by a node. In other words, the files are only available if at least one machine stores them. Therefore, we have two options to store the files in IPFS: (a) create a node and maintain it or (b) use an infrastructure provider, such as Infura, Pinata, Fleek, etc.

We tried to use Pinata and Infura, but these services are limited when using the IPFS functions and methods. On the other hand, Fleek makes many functions available regarding the IPFS. However, Fleek charges for storage (with a limited free version and a considerably expensive paid version), while Infura and Pinata charge for the number of requests. Therefore, to aggregate more implementation flexibility, minimize the costs (mainly scalability), and secure the availability of files, we decided to maintain an IPFS node integrated with our server. To this end, we instantiated only one node and planned to create new nodes further to improve availability.

### 6.3.9 Implications for software developers

When using blockchain technology in current systems, we may face a significant barrier that limits its adoption: the technical and organizational complexity that users encounter when creating decentralized applications. However, this can be partially compensated for using low-code or no-code platforms Curty et al. (2022).

Docstone technology is accessible for developers without prior experience with blockchain, as they do not need to dive deeply into the area to develop specific functionalities. In addition, the developers do not need to manage the network infrastructure, as many activities are outsourced to the service provider. Consequently, companies and developers can obtain advantages, such as reduced costs and implementation times for complex requirements.

On the other hand, Docstone focuses on client applications that require recording different assets and tracking information rather than being a generic tool for exchanging cryptocurrencies, for example. However, considering our generalist architecture, Docstone may move towards other types of generic business rules beyond registration data.

## 7 Threats to Validity

Following Wohlin et al. (2012), we discuss the potential threats that may affect the validity of our work and which strategies we adopted to mitigate them.

Regarding *internal threats*, our blockchain test networks may exhibit latency changes at different intervals, even with minor variations. To mitigate this threat, we performed 100 runs, considering the mean and standard deviations for each type of operation. Additionally, test networks usually do not behave like the main networks. However, since executing operations on the leading network incurs actual costs due to using cryptocurrencies, test networks with remote nodes provide an alternative to simulate reality, unlike local network alternatives such as Ganache<sup>10</sup>.

Regarding *external threats*, the implemented SC is limited to clients who only wish to use the functionalities of registration, search, and validation. Nevertheless, we can achieve the generality of this application since several applications share similar principles, especially for registering information to ensure integrity and future validation. In addition, the client application can choose the type of SC and Asset that best suits its needs. Additionally, we did not fully explore the permission control resources among multiple organizations on the private network. However, we acknowledge that a single organization architecture is sufficient for a first version in the particular context of Docstone.

As for *construction threats*, additional metrics related to the internal characteristics of blockchains could have been added to complement the performance analysis, however, the metric used is sufficient for our goal in this study's version, which aims to compare the different types of SC, assets, and blockchains' performance of two new blockchains (Sepolia and Alfajores) integrated into Docstone.

Regarding *conclusion threats*, the number of requests in the evaluation may not reflect real-world environments,

<sup>10</sup> <https://trufflesuite.com/ganache/>



especially when dealing with an API that should support more than one type of client application. Nevertheless, the goal was to understand the blockchains' behavior, the API's availability, and the difference between types of SC and Assets.

## 8 Conclusion

Document management has been highly relevant given the speed and quantity in which information is generated, transferred, and shared through resources and processes to ensure the necessary quality requirements. Considering such particularities, the use of blockchain shows promise because of the unique ability to record digital events transparent, securely, and resiliently immutably. Still, there may be barriers to blockchain development, given the difficulty of deployment and the high operation and maintenance cost. To this end, using "Blockchain-as-a-Service" can reduce efforts by enabling infrastructure deployment and monitoring the blockchain network in the cloud. However, they can become extensive in simple applications that only require functionalities in blockchain, such as recording, searching, and validating document information, which has been quite common in applications.

This paper extended previous studies which proposed a solution that allows the customization of parameters from the creation of templates of documents to using a specific blockchain, among some options. As new contributions, there are 1) a discussion about a software architecture adaptation of Docstone, enabling it to support new assets and smart contracts in a versatile and personalized manner that caters to other use cases, such as process tracking and asset tokenization via NFTs; 2) implementation of a mechanism that automatically generates IPFS directories, which are mapped to each smart contract, resulting in orderly storage of a variety of media formats related to each asset type. 3) a comprehensive analysis of design decisions for decentralized applications, and 4) an evaluation of several asset types to compare the execution performance of different smart contract codes and blockchains.

This extension shows that NFTs have higher deployment latency times than Documents in Sepolia and Alfajores. However, Documents have the lowest latency time in both networks for writing and reading requests, likely due to their more straightforward SC implementation. The difference in SC implementation impacts more than the types of assets themselves. In addition, we figured out that Alfajores presented a better performance in most of the four analyses. The complexity of SCs affects the cost of blockchain fees and the impacts from the asset depending on the content inserted.

In future work, we intend to implement SCs with new features to allow customers to select more specific rules and logic besides registration, search, and validation. Furthermore, we planned to conduct new experiments to evaluate the solution in a scenario of parallel requests from

different organizations to test the API's robustness when exposed to requests from different sources. Additionally, we intended to apply specific methods for evaluating software architectures, such as the Architecture Trade-off Analysis Method (ATAM), to determine the extent and potential of Docstone in meeting its expected quality attributes from an architectural and business perspective.

## References

- Ab Aziz, A., Yusof, Z. M., Mokhtar, U. A., and Jambari, D. I. (2020). The implementation guidelines of digital document management system for malaysia public sector: expert review. *International Journal on advanced science engineering information technology*, 10(1):198–204.
- Abreu, A. W. S., Coutinho, E. F., and Bezerra, C. I. (2020). A blockchain-based architecture for query and registration of student degree certificates. In *Proceedings of the 14th Brazilian Symposium on Software Components, Architectures, and Reuse*, pages 151–160.
- Acharya, V., Yerrapati, A. E., and Prakash, N. (2019). *Oracle Blockchain Quick Start Guide: a practical approach to implementing blockchain in your enterprise*. Packt Publishing Ltd, Birmingham.
- Afrianto, I. and Heryanto, Y. (2020). Design and implementation of work training certificate verification based on public blockchain platform. In *2020 Fifth International Conference on Informatics and Computing (ICIC)*, pages 1–8. IEEE.
- Bandara, E., Liang, X., Shetty, S., Ng, W. K., Foytik, P., Ranasinghe, N., Zoysa, K. D., Langöy, B., and Larsson, D. (2020). Lekana-blockchain based archive storage for large-scale cloud systems. In *International Conference on Blockchain*, pages 169–184. Springer.
- Bashir, I. (2017). *Mastering blockchain*. Packt Publishing Ltd, Birmingham, United Kingdom.
- Beck, R., Avital, M., Rossi, M., and Thatcher, J. B. (2017). Blockchain technology in business and information systems research.
- Benet, J. (2014). Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*.
- BNBChain (2022). Bnb chain documentation.
- Buterin, V. et al. (2013). *Ethereum White Paper*, volume 1. GitHub repository, Londres, Inglaterra.
- Christidis, K. and Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303.
- Curty, S., Härer, F., and Fill, H.-G. (2022). Blockchain application development using model-driven engineering and low-code platforms: A survey. In *Enterprise, BusinessProcess and Information Systems Modeling: 23rd International Conference, BPMDS 2022 and 27th International Conference, EMMSAD 2022*,

- Held at CAiSE 2022, Leuven, Belgium, June 6–7, 2022, *Proceedings*, pages 205–220. Springer.
- Das, M., Tao, X., Liu, Y., and Cheng, J. C. (2022). A blockchain-based integrated document management framework for construction applications. *Automation in Construction*, 133:104001.
- de Oliveira Melo, C. M. and Neto, J. A. M. (2014). Sistemas automatizados: discussões acerca de seus benefícios para as unidades de informação. *HOLOS*, 1:152–169.
- Eberhardt, J. and Tai, S. (2017). On or off the blockchain? insights on off-chaining computation and data. In *European Conference on Service-Oriented and Cloud Computing*, pages 3–15. Springer.
- Gaur, N., Desrosiers, L., Ramakrishna, V., Novotny, P., Baset, S. A., and O’Dowd, A. (2018). *Hands-On Blockchain with Hyperledger: Building decentralized applications with Hyperledger Fabric and Composer*. Packt Publishing Ltd.
- Greve, F. G. et al. (2018). Blockchain e a revolução do consenso sob demanda. In *Anais...*, pages 1–52, São Paulo, Brasil. Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), 36., 2018, São Paulo, SBC.
- Guide, R. M. (2021). Solidity lang.
- Han, J., Kim, H., Eom, H., and Son, Y. (2021). A decentralized document management system using blockchain and secret sharing. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 305–308.
- Jie, S., ZHANG, P., ALKUBATI, M., Yubin, B., and Ge, Y. (2021). Research advances on blockchain-as-a-service: Architectures, applications and challenges. *Digital Communications and Networks*.
- Kim, H. (2020). *Digital Document Management System with Distributed Permission Using Secret Sharing Scheme*. PhD thesis, Seoul National University Graduate School.
- Klaytn (2022). Position paper - klaytn.
- Koteska, B., Karafiloski, E., and Mishev, A. (2017). Blockchain implementation quality challenges: a literature. In *Anais...*, pages 11–13, Belgrade, Serbia. Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), 6., 2017, Belgrade, Serbia, CEUR.
- Li, X., Zheng, Z., and Dai, H.-N. (2021). When services computing meets blockchain: Challenges and opportunities. *Journal of Parallel and Distributed Computing*, 150:1–14.
- Macedo, A. J., Araújo, A. A., and Taveira, I. (2021). Adoção de blockchain para apoio ao cadastro e inspeção de barragens hídricas: Uma proposta de pesquisa baseada em design science research. In *5ª Conferência sobre Sistemas de Informação na América Latina (ISLA)*.
- Martiri, E., Muca, G., Xhina, E., and Hoxha, K. (2018). Dmsxt: A blockchain-based document management system for secure and intelligent archival. In *RTA-CSIT*, pages 70–74.
- Morais, S. C. B., Mussi, C. C., and de Lima, M. A. (2021). Tecnologia da informação e desempenho da gestão documental: uma estrutura conceitual. *Revista Brasileira de Preservação Digital*, 2:e021004–e021004.
- Navedrov, D. (2006). Using jmeter to performance test web services. *Published on dev2dev*, pages 1–11.
- Onik, M. M. H. and Miraz, M. H. (2019). Performance analytical comparison of blockchain-as-a-service (baas) platforms. In *International Conference for Emerging Technologies in Computing*, pages 3–18. Springer.
- Saraiva, R., Araújo, A. A., Soares, P., and Souza, J. (2021). Miriam: A blockchain-based web application for managing professional registrations of medical doctors in brazil. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–2. IEEE.
- Shukla, P. A. and Samet, S. (2020). Systematization of knowledge on scalability aspect of blockchain systems. In *Anais...*, pages 130–138. Future of Information and Communication Conference, 2020, San Francisco, United States, Springer.
- Soares, P., Saraiva, R., Fernandes, I., Neto, A., and Souza, J. (2022a). A blockchain-based customizable document registration service for third parties. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–2. IEEE.
- Soares, P., Saraiva, R., Fernandes, I., Souza, J., and Loiola, R. (2022b). Docstone: A blockchain-based architecture for a customizable document registration service. In *Proceedings of the 16th Brazilian Symposium on Software Components, Architectures, and Reuse*, pages 1–10.
- Stefanović, M., Pržulj, Đ., Ristić, S., Stefanović, D., and Nikolić, D. (2022). Smart contract application for managing land administration system transactions. *IEEE Access*, 10:39154–39176.
- Technology, P. (2022). Polygon documentation.
- Tikhomirov, S. (2017). Ethereum: state of knowledge and research perspectives. In *International Symposium on Foundations and Practice of Security*, pages 206–221. Springer.
- Van Mölken, R. (2018). *Blockchain across Oracle: understand the details and implications of the Blockchain for Oracle developers and customers*. Packt Publishing Ltd.
- Wan, Z., Cai, M., Yang, J., and Lin, X. (2018). A novel blockchain as a service paradigm. In *International Conference on Blockchain*, pages 267–273. Springer.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media, Boston.
- Wöhler, M., Zdun, U., and Rinderle-Ma, S. (2021). Architecture design of blockchain-based applications. In *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 173–180. IEEE.
- Xie, J., Yu, F. R., Huang, T., Xie, R., Liu, J., and Liu, Y. (2019). A survey on the scalability of blockchain systems. *IEEE Network*, 33(5):166–173.
- Xu, X., Weber, I., and Staples, M. (2019a). *Architecture for blockchain applications*. Springer.

- Xu, X., Weber, I., and Staples, M. (2019b). Blockchain patterns. In *Architecture for Blockchain Applications*, pages 113–148. Springer.
- Zheng, Z., Xie, S., Dai, H., Chen, X., and Wang, H. (2017). An overview of blockchain technology: Architecture, consensus, and future trends. pages 557–564, Boston. IEEE international congress on big data (BigData congress), 1., 2017 Boston, IEEE.