AuditChain: Blockchain views applied to auditing

Afonso Martins Marques

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. André Ferreira Ferrão Couto e Vasconcelos
Prof. José Eduardo Pina Miranda

Examination Committee

Chairperson: Prof. Nuno Miguel Carvalho dos Santos
Supervisor: Prof. André Ferreira Ferrão Couto e Vasconcelos
Member of the Committee: Prof. Rolando da Silva Martins

November 2022

Acknowledgments

From the very first moment I want to thank all of the people that somehow influenced the development of the thesis, without you it would have been much more difficult.

- Professor André Ferreira Ferrão Couto e Vasconcelos, Professor José Eduardo Pina Miranda, my dissertation supervisors, for their perseverance, support, guidance and sharing of knowledge that made this thesis possible.
- Rafael Belchior for all the patience, advice, clarifications, and support throughout the development, you were undoubtedly one of my greatest inspirations.
- My family João, Sandra, Eduardo, José, Carlos, Maria O., Maria E., Claudia and Orlando for their support, help, freedom, happiness, trust and love that they have offered me throughout my life.
- My friends and colleagues for all the moments we shared throughout our academic journey, without you it would have been a much more difficult path.
- Slavisa Tomic for the extra motivation to keep searching for more knowledge and to always believe in what I will be able to achieve.
- Pedro Serra for his support and advices that helped me improve as a student and as a person.

To all of you my deepest gratitude.

Abstract

Blockchain, an emerging technology, provides a decentralized, and immutable data storage, replicated across untrusting peers. Contrary to common knowledge, private blockchain stakeholders can have different views on the same distributed ledger, and thus several challenges are raised. This happens because as soon as complexity increases, different representations are possible and this way different business process views are created on the same blockchain. Currently, it is hard to visualize data partitions within a blockchain and perform arbitrary operations on it, for instance merging two blockchain views to conduct a blockchain migration or audit. In this document, we will explain the concept of view and how it can be used for several purposes. We also present the entire process required to obtain the data and subsequently create the blockchain view. We will likewise identify the possible mechanisms and the tool created *BUNGEE*, for this purpose. Subsequently, to the creation of the blockchain view, various stakeholders can prove its internal state, contributing to blockchain interoperability, and enhanced the way audits are performed. This thesis aims to contribute to blockchain interoperability and easing of blockchain audits by leveraging the concept of blockchain view.

Keywords

Blockchain audit, Blockchain View, Interoperability, BUNGEE, AuditView

Resumo

Blockchain, uma tecnologia emergente, proporciona um armazenamento de dados descentralizado, imutável e replicado entre utilizadores confiáveis e não confiáveis. Ao contrário do conhecimento comum, os intervenientes de Blockchain privadas podem ter diferentes *views* sobre a mesma *ledger*. Desta forma, vários desafios são levantados. Isto acontece dado que com o aumento da complexidade, é possível fazer diversas representações, criando desta forma diferentes *views* do processo de negócio na mesma blockchain. Atualmente é difícil visualizar partições de dados dentro de uma blockchain e realizar operações arbitrárias sobre a mesma, por exemplo, unir duas *views* de forma a conduzir uma migração da blockchain ou fazer auditoria. Neste documento, explicaremos o conceito de *view*, e como este pode ser utilizado para diversas finalidades. Apresentamos também todo o processo necessário para a obtenção dos dados e posteriormente a criação da blockchain *view*. Identificaremos da mesma forma os possíveis mecanismos e a ferramenta criada *BUNGEE*, para tal. Posteriormente à criação da blockchain *view*, vários stakeholders podem provar o seu estado interno, contribuir para a interoperabilidade da blockchain e melhorar a forma como as auditorias são efetuadas. Esta tese tem como finalidade apoiar o processo de auditorias em blockchains, potenciando o conceito de *view*.

Palavras Chave

Auditoria na Blockchain, Blockchain *view*, Interoperabilidade, BUNGEE, AuditView

Contents

List of Figures

List of Tables

Listings

1 Introduction

In recent years the use of blockchain technology has increased exponentially, a similar thing has happened with the development of new tools and ways to use it, however, when it comes to the auditing aspect, the same does not happen. Several studies have been done on the advantages and importance of blockchain auditing, although there are not many tools available to do auditing, and those that do exist, many are just theoretical solutions and the rest are very limited in terms of functionalities, being only useful when the subject is Cryptocurrencies. This area of blockchain auditing is a recent topic, however it has already sparked interest from quite a few eager investors to be the first to invest in a pioneering solution.

In this document we will introduce what is blockchain technology and its types, we will also introduce the concept of blockchain View. We are going to explain what auditing is, its use cases and also blockchain auditing. We will identify the current problem and a possible pioneering solution for it. We will present all the implementation decisions regarding the tools developed. We will also talk about the reason that led us to use certain technologies over others. It will be presented how each one of them was evaluated and the tests used on each.

Throughout the document we will refer to blockchain View several times, however it will be presented as View for simplicity.

2 Background and Related Work

Contents

In this section, we will present and explain the foundation stones of this project. We also present work that has been done on blockchain and blockchain audit in order to bridged those and the proposed problem. Since there are no solutions directly related to the tool we propose to develop, we have chosen to combine the topic background and relatedwork.

2.1 Blockchain

Blockchain is a technology firstly introduced in 2008 by a person or a group with the pseudonymous *Satoshi Nakamoto* after the realise of bitcoin whitepaper[1]. This technology has (slowly) become one of the most frequently discussed methods for securing data storage and transfer through decentralized, trustless, peer-to-peer systems [1]. The blockchain technology provides a decentralized, open, Byzantine fault tolerance transaction mechanism, and promises to become the infrastructure for a new generation of Internet interaction, including anonymous online payments, remittance, and transaction of digital assets [2].

The idea behind this technology is relatively simple. Blockchain technology is constituted by sequential blocks linked together, thus forming a chain. Each block contains a specific group of information (records), constituted by a timestamp and a hash of the previous block in the chain. The first block is called the genesis block, it differs from the others, since it has no block before and thus no reference.

Hash is a cryptographic digest, produced by a hash function. This function has some important characteristics such as, one way (non-invertible), strong collision resistance (computationally infeasible to find two inputs that give the same hash). These characteristics are important since we validate the integrity of the blocks using the hash. The validation process ensures that the records are valid and therefore the block. After this step, the block will be added if the participants reach a consensus. Consensus is reached by participants of the network by a consensus algorithm.

Participants also called miners compete in order to generate the next accepted block. This competition consists of solving a computationally heavy mathematical problem in the most efficient way possible. During this competition, in case of more than one miner solve the hash puzzle at the same time, the blockchain may "fork" leading to parallel chains. Such a scenario is eventually resolved by the miners picking the longest chain [3, 4]. This competitions is associated with Prof-Of-Work (*POW*) which is the consensus mechanism for Bitcoin, the first application of the blockchain technology [5]. After the miners finish their tasks (generate the next block), they are rewarded, this reward depends on the blockchain [2]. The problem difficulty varies during the time, it is adjusted in order to keep the block generation pace constant, e.g. More participants means more miners

[1] https://bitcoin.org/bitcoin.pdf

competing for the next block, so more blocks would be generated per unit of time, to avoid this, the difficulty of the problem is increased. Naturally, the opposite happens with fewer participants.

Blockchain was plenty of advantages compared with a centralized technology, however is not untouchable. There are a few known attacks, such as: Distributed denial of service (*DDoS*), *51%-Attack*, *Race attack*, *Sybil attack*, among many others.

The first attack, *DDoS* is hard to perform on the blockchain network because of their nature, however is not impossible. When this attack is performed, the intention is to bring down a server by consuming all its processing resources with numerous requests. DDoS attackers aim to disconnect a network's mining pools, e-wallets, crypto exchanges, and other financial services. A blockchain can also be hacked with DDoS at its application layer using DDoS botnets.

An attack scenario against the consensus mechanism is called the *"51% attack."* In this scenario, a group of miners, controlling a majority (51%) of the total hash power of the network, conspire to attack. With the ability to mine most blocks, attacking miners can spawn deliberate bifurcations in Blockchain, generate double-spend transactions, or perform denial of service attacks (DoS) against specific addresses or transactions. A bifurcation attack or double-spend attack is an attack where the attacker causes already confirmed blocks to be invalidated by bifurcating a level below them, with a later reconvergence in an alternate chain. With enough power, an attacker can invalidate six or more blocks in a sequence, invalidating transactions that were previously considered immutable (with six acknowledgments). Note that double spending can only be done on the attacker's transactions, for which the attacker can produce a valid signature. Making a double spend of the transaction itself is profitable when, by invalidating a transaction, the attacker can receive an irreversible payment or product without having to pay for it [6].

Race attack happen when an attacker sends two conflicting transactions in rapid succession into the network. This type of attack is relatively easy to implement in PoW-based blockchains. Merchants who accepts a payment immediately with 55 "0/unconfirmed" are exposed to the transaction being reversed. There are some possible countermeasures: "Listening period", "Inserting observers" and "Forwarding double spending attempts" [7].

In a public blockchain we can have a *sybil attack* where one hostile peer create lots of fake identities in order to defraud the system to break its trust and redundancy mechanism. [8].

2.1.1 Public or Permissionless Blockchain

Permissionless means that anyone may join or leave the network at will. Public means that anyone, in principle, may propose a new state of the ledger [9]. Permissionless blockchain networks are decentralized ledger platforms open to anyone publishing blocks, without needing permission from any authority. Permissionless blockchain platforms are often open source software, freely available to anyone who wishes to download them. Since anyone has the right to publish blocks, this results in the property that anyone can read the blockchain as well as issue transactions on the blockchain (through including those transactions within published blocks). Any blockchain network user within a permissionless blockchain network can read and write to the ledger. Since permissionless blockchain networks are open to all to participate, malicious users may attempt to publish blocks in a way that subverts the system. To prevent this, permissionless blockchain networks often utilize a multiparty agreement or 'consensus' system that requires users to expend or maintain resources when attempting to publish blocks. This prevents malicious users from easily subverting the system. Examples of such consensus models include proof-of-work and proof-of-stake methods. The consensus systems in permissionless blockchain networks usually promote non-malicious behavior through rewarding the publishers of protocol-conforming blocks with a native cryptocurrency [10].

Proof-of-stake (PoS) aims to replace the way of achieving consensus in a distributed system. Instead of solving the Proof-of-Work, the node which generates a block has to provide a proof that it has access to a certain amount of coins before being accepted by the network. Generating a block involves sending coins to oneself, which proves the ownership. The required amount of coins (also called target) is specified by the network through a difficulty adjustment process similar to *PoW* that ensures an approximate, constant block time. [11]

2.1.2 Private or Permissioned Blockchain

Similarly to the public blockchain, this one is also constituted by connected blocks, however it has some particularities in what concerns its members and the actions they can perform. It is considered private or permissioned since its participants are restricted to a set of permitted actions. External participants cannot submit or participate in transaction validation process [12].

In this type of blockchain users are added by a person/authority (be it centralized or decentralized), and it is not possible to join spontaneously. This type of blockchain is usually used when all its participants/organizations are known, but not all of them can be trusted. It is also used by organizations that need to more tightly control and protect their blockchain. These characteristics pose the problem of blockchain users having to trust on a single entity

Permissioned blockchain networks may thus allow anyone to read the blockchain or they may restrict read access to authorized individuals. This blockchain may be instantiated and maintained using closed or open source software. The consensus model can be determined by the organizations, based on how much they trust one another.Beyond trust, permissioned blockchain networks provide transparency and insight that may help better inform business decisions and hold misbehaving parties accountable. This can explicitly include auditing and oversight entities making audits a constant occurrence versus a periodic event [11].

2.2 Hyperledger Cactus

Hyperledger Cactus[2] is a framework for integrating distributed ledgers. This tool was build under Hyperledger ecosystem aiming to provide decentralized, secure and adaptable integration between blockchain networks. One of the reasons why Cactus was created was the need to suppress an existing lack of frameworks/tools that would allow the integration of different environments, technologies and systems within a company or even among several companies, thus facilitating business. This framework is relatively new, however it already provides a pluggable architecture that enables the execution of ledger operations across blockchains, allowing users to make integration between different blockchains.

2.3 Audit

Before we explain what auditing is, it is important to note that there are several types of auditing [13]. When most people think about audit, they imagine IRS agents arriving in an unannounced way, calculators in hand, ready to analyze countless boxes of receipts, invoices and bills. That is one type of audit, but it is certainly not the only one. Most audits have nothing to do with tax day and none of them are unannounced.

Traded companies are required by the securities and exchange commission to validate their financial position with an audit [14], and although they're not legally required to, privately held companies often perform audits at the request of banks, investors, and other key stakeholders. The goal is to ensure investors and other stakeholders that their cashflows balance sheets and profit and loss statements aren't materially misstated.

Public or private companies that go through an audit have the opportunity to gain valuable insight into how their businesses are performing [15].

[2]https://www.hyperledger.org/use/cactus

The word audit means, to evaluate. Auditors are the ones who evaluate where money / information is coming from, where it's going, and what it's doing each step of the way.

Auditors need to think critically in order to understand what business decision drive the transactions. They also have to verify if what the company states is correct. During the audit process, they compare the documents the company uses in their day-to-day operations against what they have recorded in their financial statements.

Audit engagement can take between couple mouths till one year, depending on the size of the client and the complexity of the project.

2.4 Blockchain Audit

Blockchain auditing has the same objective as auditing, but is done differently, taking advantage of the characteristics of the blockchain [16]. As explained before, blockchain is essentially a type of database known as a distributed ledger that is decentralized with no central administrator. When a "user" saves a transaction in the database it is timestamped and saved in a block. Each "user" keeps a copy of the distributed ledger, the data is replicated and synchronized between all copies of the ledger in real-time. Since the ledger is both shared and encrypted, an attacker who wants to change the data contained in it, would have to simultaneously hack each node of the network and overcame the encryption [17], even if the attacker succeeded, users would be able to identify which data was tampered with, thus eliminating one of the disadvantages of traditional systems, single point of failure. This high level of security is one of the main attractions of the blockchain. It is increasingly being used for a range of functions where it is important to transmit data securely, particularly within the transaction-based financial services industry. For example block blockchain technology can be used to process payments, to create verifiable audit trails and to register digital assets such as stocks and bonds. In fact blockchain as the potential to change the way business in every sector operate. Once data stored in distributed ledgers is continually updated it offers finance team the possibility of real-time [18] reporting to both management and external auditors. This could free up auditor to offer more value-added services to their clients, such as strategic planning and support with wider business decisions. Like any technology, this one also has some drawbacks including the cost of implementation, privacy issues, lack of agreed standards and a limited scalability of blockchains.

2.5 Blockchain View

This concept of blockchain view is relatively new and there is very little related work. At this moment there are not many documents that use this concept yet, however we believe that in the future it will be widely used since it brings clarifications that were missing until today.

Blockchain views are tools that promote data portability once they enable blockchain interoperability. A view is an intermediary standardized data format not dependent on a specific blockchain (agnostic) that can be translated across blockchains. Views are self describing because they are represented via a multi-hash. They denote the commitment (e.g., via accumulators or Merkle trees) of a state in a particular time from a stakeholder centric perspective [19].

2.6 BUNGEE

BUNGEE (Blockchain UNifier view GEnErator) is a view generator, a system composed of four software components that creates, and processes views from a set of views, each one from a specific stakeholder. BUNGEE constructs views from a set of states coming from an underlying blockchain, giving them as input to a controller. The controller component can process views (including merging views or other processing after a view is merged). BUNGEE abstracts both analysts and developers from the underlying data structures from blockchains, and allow them to rationalize about the collected data from a business perspective [19].

3 Hyperledger Fabric

Contents

Hyperledger Fabric[1] is an enterprise-grade, distributed ledger platform that offers modularity and versatility for a broad set of industry use cases. The modular architecture for Hyperledger Fabric accommodates the diversity of enterprise use cases through plug and play components, such as consensus, privacy and membership services.

On Hyperledger network, only parties directly related to the transaction deal are updated on the ledger, thus maintaining privacy and confidentiality. Hyperledger primarily leverages the concept of *channels* to ensure privacy and confidentiality. A *channel* is a virtual blockchain network which sits on top of a physical blockchain network and has its own access rules. They employ their own mechanism for transaction ordering and thus ensure scalability, thereby allowing effective ordering along with separation of data.

Chaincode[2] is a program, written in Go, node.js, or Java that implements a prescribed interface. It runs in a secured Docker container isolated from the endorsing peer process. Chaincode initializes and manages ledger state through transactions submitted by applications. Typically, it handles business logic agreed to by members of the network, so it may be considered as a "smart contract". State created by a chaincode is scoped exclusively to that chaincode and can't be accessed directly by another one. However, within the same network, given the appropriate permission a chaincode may invoke another to access its state. To deploy a chaincode, a network admin must install the chaincode onto target peers and then invoke an orderer to instantiate the chaincode onto a specific channel. While instantiating the chaincode, an admin can define an endorsement policy to the chaincode.

Endorsement policy defines which peers need to agree on the results of a transaction before the transaction can be added onto ledgers of all peers on the channel.

Peer is a blockchain node that stores all transactions on a joining channel. Each peer can join one or more channels as required. However, the storage for different channels on the same peer would be separate. Therefore, an organization can ensure that confidential information would be shared to only permitted participants on a certain channel.

Orderer is one of the key elements employed in the the Fabric consensus mechanism. Orderer is a service responsible for ordering transactions, creating a new block of ordered transactions, and distributing a newly created block to all peers on a relevant channel.

Certificate Authority or CA is responsible for managing user certificates such as user registration, user enrollment, user revocation. More specifically, Hyperledger Fabric is a permissioned blockchain network that uses an X.509 standard certificate to represent permissions, roles, and attributes to each user.

Client is considered to be an application that interacts with Fabric blockchain network. *World*

[1] https://www.hyperledger.org/use/fabric
[2] https://hyperledger-fabric.readthedocs.io/en/release-1.3/chaincode.html

State maintains the current state of variables for each specific chaincode.

Hyperledger Fabric currently can support two types of World State database including *LevelDB* and *CouchDB*. LevelDB is a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values. CouchDB is a JSON-based database supporting rich querying operations based on JSON objects. The following figure represents a Fabric network with chaincodes and ledger attached.

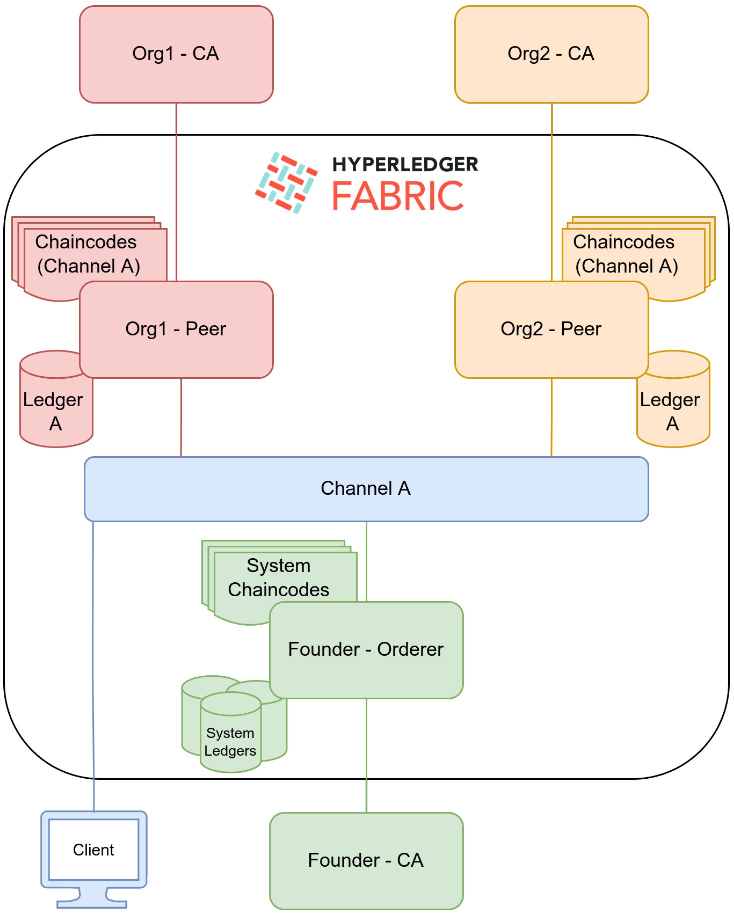


Figure 3.1: Hyperledger Fabric Network

In the previous figure 3.1 as we can see, Org1's Peer and Org2's Peer mutually join the same channel. There can be multiple chaincodes instantiated on the same channel. Furthermore, the instantiated chaincode can be upgraded if needed. This makes a chaincode to be updatable or patchable.

Hyperledger Fabric framework was contributed by IBM, Fabric works as a foundation for developing applications with a modular architecture. The framework utilizes container technology to host smart contracts, chaincode comprising the systems applications logic.

3.1 Hyperledger Fabric Ledger Creation

In order to create a Hyperledger Fabric ledger we have to follow the nine steps represented in the figure 3.2

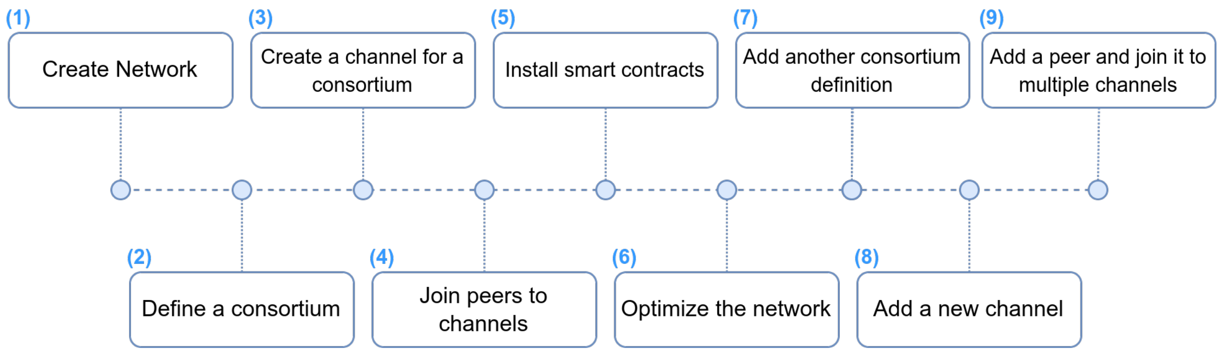


Figure 3.2: Hyperledger Fabric Creation Steps

1. *Network creation*, this network is created from the definition of the consortium including its clients, peers, channels and ordering service. The ordering service is the administration point for the network because it contains the configuration for the channel within the network.

2. *Consortium definition*, this consortium is composed of two or more organizations on the network. Consortiums are defined by organizations that have a need for transacting business with one another, and they must agree to the policies that govern the network.

3. *Channel creation* for a consortium, channel is a mean of communication used to connect the components of the network and or the member client applications.

4. *Join peers to channels*, peers are joined to channels by the organizations that own them and there can be multiple peer nodes on channels within the network.

5. *Install smart contracts*, smart contract, chaincode must be installed and instantiated on a peer in order for a client application to be able to invoke the smart contract.

6. *Optimize the network*, while there is no theoretical limit on how big a network can get as the as the network grows it is important to consider design choices that will help optimize network throughput stability and resilience.

7. *Add another consortium definition*, as consortium are defined and is added to the existing channel, we must update the channel configuration by sending a channel configuration update transaction to the ordering service.

8. *Add a new channel*, organizations are what form and join channels, and channel configurations can be amended to add organizations as the network grows. When adding a new channel to the network, channel polices remain separate from other channels configured on the same network.

9. *Add a peer and join it to multiple channels*, we can continue adding new peers and connect them to several channels within the network.

4

Hyperledger Cactus

Contents

Hyperledger Cactus[1] is a framework for integrating distributed ledgers. This tool was build under Hyperledger ecosystem aiming to provide decentralized, secure and adaptable integration between blockchain networks. One of the reasons why Cactus was created was the need to suppress an existing lack of frameworks/tools that would allow the integration of different environments, technologies and systems within a company or even among several companies, thus facilitating business. This framework is relatively new, however it already provides a pluggable architecture that enables the execution of ledger operations across blockchains, allowing users to make integration between different blockchains.

4.1 Use Cases

To get a better understanding of Hyperledger Cactus we will present six use cases as represented in figure 4.1.

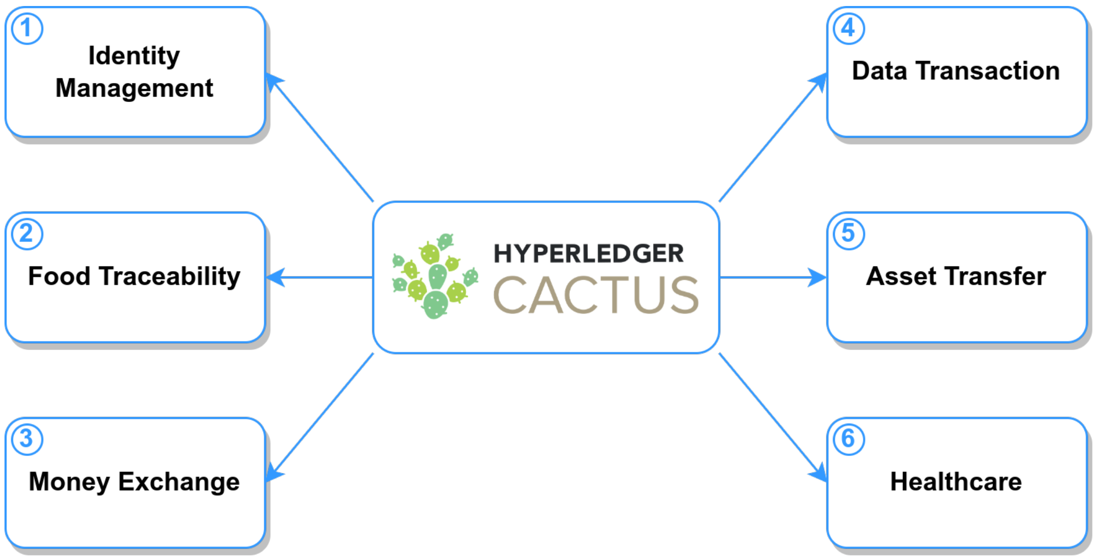


Figure 4.1: Hyperledger Cactus Use Cases

Identity Management, users have the ability to manage and interact with wallets across different permissionless and permissioned ledgers, thereby allowing users to connect different networks through a single interface.

Food Traceability, blockchain technology has made progress when it comes to food integration. One organization that has made a great effort to contribute to the area of food traceability is the IBM Food Trust by providing a smarter, safer, and sustainable environment. Hyperledger Cactus offers the possibility to complement this environment by providing means of the software implementation project. Retailers by integrating Cactus as an architectural component and providing an interface

[1] https://www.hyperledger.org/use/cactus

to the consumer, they allow them to trace food during the supply chain, allowing consumers to see the origin, shipping and storage of the products, and therefore being able to make a purchase with as much information as possible.

Money Exchange, Hyperledger Cactus is also effective when it comes to pegging stable coins against other cryptocurrencies. In this use case, one user can actually implement and use Hyperledger Cactus to set an environment and use the necessary plugins to make the ledger work for token minting, transactions, and burning.

Data Transaction, Hyperledger Cactus can facilitate peer-to-peer data/document transaction.

Asset Transfer, Hyperledger Cactus allows to transfer funds from one ledger to another, offering escrowed asset transfer social interaction. This interaction is important as it will give the user the flexibility to choose the blockchain ledger of his choice.

Healthcare, this industry has been struggling with data sharing. The use of blockchain technology, however, enables them to share data among themselves. Nevertheless, the use of different blockchain technology means the need for proper data exchange and interoperability. Hyperledger Cactus can help solve the problem with the help of peer to peer data sharing social interaction.

4.2 Hyperledger Cactus architecture

Hyperledger Cactus has a complex architecture, however it is quite easy to grasp once we understand each component that constitutes it. The figure 4.2 is a visual high level representation of the architecture.

This architecture is composed by three main components, being them, *Ledger Plugin*, *Business Logic Plugin* and *Routing Interface*.As we can verify, it all comes down to plugins.

Business Logic Plugin is a generic placeholder for any code we intend to operate in order to add value for our use case. This plugin can be considered the kernel/core of cactus and is delivered by the maintainers. In the whole architecture, there must be only one plugin with these characteristics.

Routing Interface is still being designed, but the main purpose is to make it as seamless as possible in order to send our API requests to a specific DNS host on the internet which is cactus deployment and still hit a particular plugin that we are looking to work with.

Ledger plugin are responsible for knowing and understanding how to communicate directly with the ledger without the need to be configured from scratch. As regards to this component, there may be several in the architecture. At the moment, there are plugins developed for the following ledgers, Hyperledger Besu, Corda, Hyperledger Fabric, Ethereum, Hyperledger Iroha, Quorum, Hyperledger Sawtooth and Xdai.

The first step as we can see in the figure 4.2 is an API call between the user and the all de-

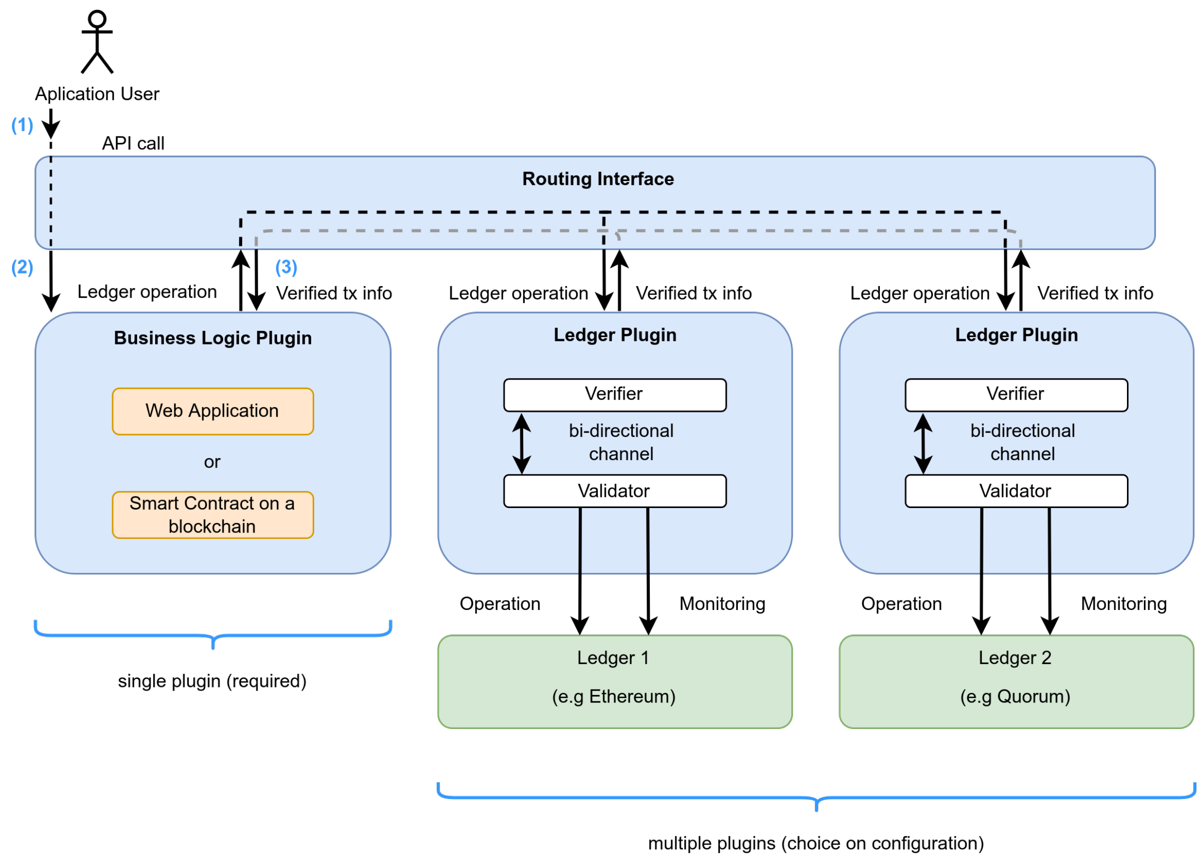


Figure 4.2: Hyperledger Cactus Architecture

ployment, it seems simple in the diagram, however it takes a great effort to get the client API in several available languages. This is especially important if we are developers writing an application where the backend is Cactus, this way we get great developer convenience since we have the client libraries ready. This means we just need to install them, perform some sort of authentication, and then we can execute transactions on different ledgers using our language of choice.

Some architectural decisions, the code was written in typescript and it gets bundled with that back so that we have the ability to have universal packages where the same code written in typescript can run on the backend and on the client side. This enables developers to go a little faster and be more productive and thus make less mistakes since there are less code duplication. Another major decision was test automation, everything has to be tested automatically on continuous integration and on every single commit that gets pushed, so that if there are any mistakes they can be traced early on. This measure was adopted since it would be practically impossible to do this validation on each component manually, given that there are numerous use cases and scenarios. Regarding plugin architecture, since the technology is constantly evolving it is not possible to predict the future, so possibly there will be new designs in the future ledgers. Hyperledger cactus developers believe that they can prepare for the unknown, developing software that bends and not breaks when major technological shifts occur. Every new Cactus plugin project can live outside the main cactus repository since it is possible to install them as a npm package, for that reason anyone can create a new plugin implementation.

5

BUNGEE

Contents

The acronym BUNGEE stands for Blockchain UNifier view GEnErator, and its purpose is to create and process blockchain Views. In the future BUNGEE will also merge blockchain Views.

In order to build the View, the process was divided into two main parts: taking a snapshot of the blockchain according to a specific participant *p* and constructing the View considering the desired time interval. Two important definitions that we must keep in mind in order to understand the BUNGEE tool are View and Snapshot.

Blockchain View offers a stakeholder-centric, generalizable, self-describing commitment to the state of a blockchain, allowing for representing states from different blockchains in a standardized way.

Snapshot is the state of a system in our case blockchain at a particular point in time.

5.1 View

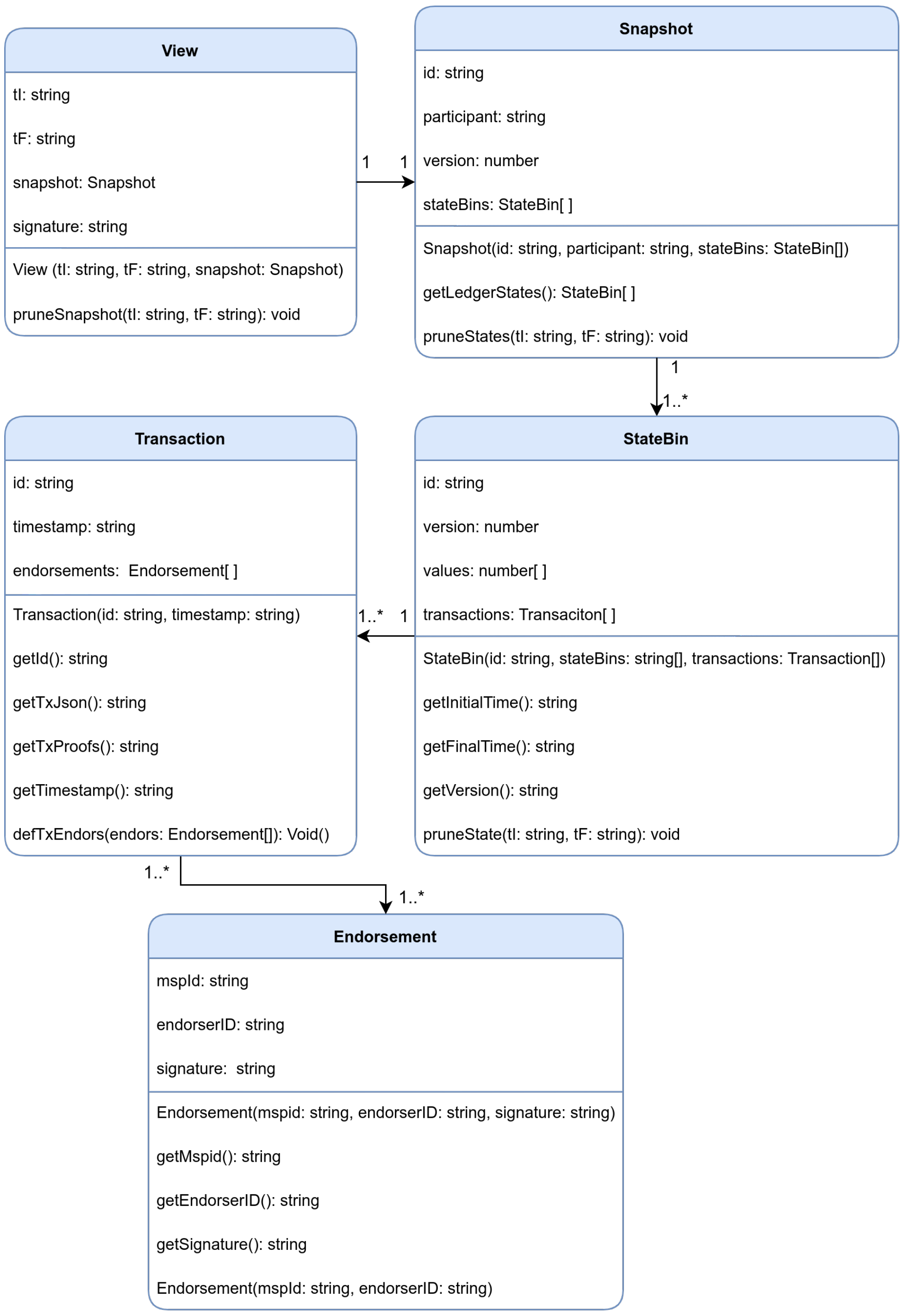
On the implementation side, five main data classes were created, as shown in figure 5.1. We will now proceed to analyze each of them in more detail.

In order to be easier to understand how these classes relate to each other, we will introduce them from the simplest to the most complex class.

Class *Endorsement* could also have been named proof, since it concerns the proof regarding the approval of a transaction. This is the simplest class, consisting of the Membership Service Providers (MSP), Id, endorser Id and finally the signature. The functions created for this class were only getters of its attributes because once the class is initialized it does not change. The MSP is used to check that the peer is allowed to endorse the transaction.

Class *Transaction* which contains all the information related to a single transaction. It has as attributes the Id, Timestamp in which the transaction occurred and a list of *Endorsements*. As functions this class has getters of its properties and the function *defineTxEndorsements()* which takes as argument a list of endorsements for this transaction which is later assigned to the variable held in the class.

Class *StateBin* is a class that represents a state of an asset present in a blockchain at a given instant of time. As attributes this class has an Id, a version that can be inferred from the number of values we have for an asset, and a list of transactions that have been performed in order to change the state of our asset. As in the previous classes, this one also has getters for its attributes. We also developed a function *prune()* that takes as arguments the start and end time to do the temporal pruning. To do the pruning we go through each of the transactions and verify if they are within the defined time interval, saving all the transactions that are within the interval in a temporary list. At the end of running through all the transactions we return to the temporary list created earlier.



View
tI: string
tF: string
snapshot: Snapshot
signature: string
View (tI: string, tF: string, snapshot: Snapshot)
pruneSnapshot(tI: string, tF: string): void
1
1
Snapshot
id: string
participant: string
version: number
stateBins: StateBin[ ]
Snapshot(id: string, participant: string, stateBins: StateBin[])
getLedgerStates(): StateBin[ ]
pruneStates(tI: string, tF: string): void
1
1..*
Transaction
id: string
timestamp: string
endorsements: Endorsement[ ]
Transaction(id: string, timestamp: string)
getId(): string
getTxJson(): string
getTxProofs(): string
getTimestamp(): string
defTxEndors(endors: Endorsement[]): Void()
1..*
1
StateBin
id: string
version: number
values: number[ ]
transactions: Transaciton[ ]
StateBin(id: string, stateBins: string[], transactions: Transaction[])
getInitialTime(): string
getFinalTime(): string
getVersion(): string
pruneState(tI: string, tF: string): void
1..*
1..*
Endorsement
mspId: string
endorserID: string
signature: string
Endorsement(mspid: string, endorserID: string, signature: string)
getMspid(): string
getEndorserID(): string
getSignature(): string
Endorsement(mspId: string, endorserID: string)

Figure 5.1: BUNGEE Class Diagram

Class *Snapshot* is constituted by Id, the participant to which the snapshot should be taken, version which will initially have the value one. This value is only changed when there is a merge between two views of the same participant. And finally a list of *StateBins*. As functions we have a getter to get the ledger states, and the function *pruneStates()* which when invoked will loop through each stateBin invoking the fuction *prune()*.

At last we have the class *View*, consisting of the initial and final timestamp that will be used in the temporal pruning, we also have the snapshot and signature of all the constituent fields of the View. We also developed the function *pruneSnapshot()* which takes as arguments the timestamps. When this is invoked, it invokes the function present in the Snapshot *pruneStates()*, this way we reduce redundancy and eliminate the need to instantiate the same variables in the class Snapshot, beyond this point we meet the definition presented in the paper [19]

5.2 Plugin-Bungee

The presented classes are very important, however they are only useful if we have the data to build the View. The classes presented in the figure 5.1 do not interact directly with the blockchain, for this purpose we created *Plugin-Bungee*. To understand how this plugin works let's take a look at the diagram 5.2.

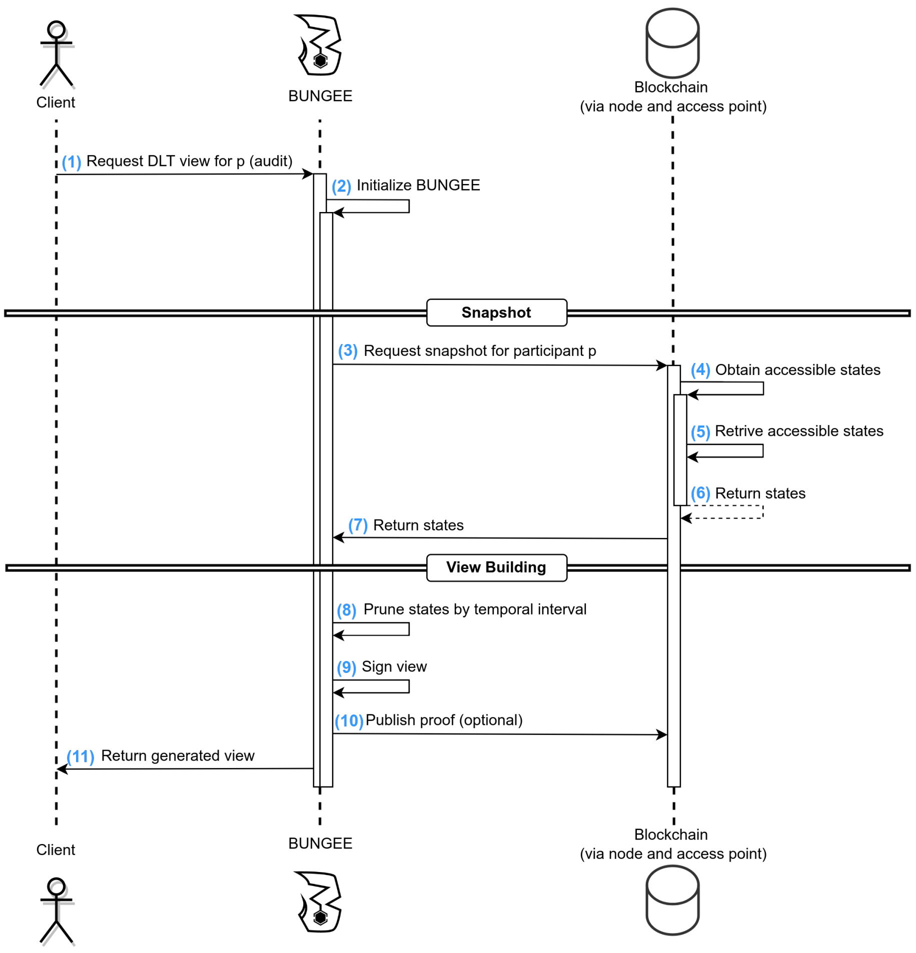

Client
BUNGEE
Blockchain
(via node and access point)
(1) Request DLT view for p (audit)
(2) Initialize BUNGEE
Snapshot
(3) Request snapshot for participant p
(4) Obtain accessible states
(5) Retrive accessible states
(6) Return states
(7) Return states
View Building
(8) Prune states by temporal interval
(9) Sign view
(10) Publish proof (optional)
(11) Return generated view
Client
BUNGEE
Blockchain
(via node and access point)

Figure 5.2: BUNGEE Diagram

The process of View creation consists of ten main steps and one optional step. In order to start the process of view generation, the client will define the participant for which the View should be generated. he will also define the time interval of the transactions he wants to analyze in the future view. After this step, he will then request blockchain view for participant *p* (1). Once the request is made, BUNGEE will initialize (2). In order to get the Snapshot for participant *p*, a request is made to the blockchain (3), then the accessible states (4) are requested for this participant, next these are obtained/retrieved (5) and finally returned (6) and sent to BUNGEE (7). From now on, there is no need to get any more data from the blockchain. After we have the snapshot, we start building the view, where we will prune states by temporal interval (8) and then all the data is signed, leaving as the final result the View (9). The step publish proof (10) is optional, however it concerns publishing the proof how the view was created on the blockchain. To finish this process, BUNGEE will return to the user the generated View (11).

The previous description was intended to give us a higher-level perspective of how the whole process of obtaining data and creating the View is done, thus giving a general idea. Turning to the implementation level, let's take a closer look at the *Plugin-bungee*. As we have seen, after the user request and the BUNGEE initialization the process can be divided into two parts, Snapshot and View building.

5.2.1 Snapshot Creation

After the client has made the blockchain view request for *p* (1) and BUNGEE (2) has been initialized, the function *generateLedgerStates()* will be invoked, this function is intended to get the data present in the blockchain in order to create the snapshot. Within this, other functions will be invoked in order to collect all the necessary data. The first function to be called is *getAllAssetsKey()* which will get all assets keys, these keys are considered the asset Ids. Once we know which assets exist, we go through each one in order to get all the transactions associated *getAllTxByKey()* with that key. In order to have the transactions for a key we need to get the information about each transaction that is associated with *p*, this way we get the receipt of each one using *fabricGetTxReceiptByTxID()*. By obtaining the receipt we will have access to information that is crucial for the view creation, these being: *transaction creator*, *asset value* and *transaction endorsements*. At this point we have almost all the information, all that is remaining is to go through each of the endorsements to get the *mspid*, *endorserId* and is *signature*. Once we have all the data that we require we then generate the ledger states. With all the states created we then invoke the function *generateSnapshot()* which will instantiate the class Snapshot shown earlier 5.1, passing as argument the Id created for the snapshot, the participant and the stateBins.

5.2.2 View Building

With the snapshot generated, the function *buildView()* is invoked. This will instantiate a View class, passing as argument the Snapshot created earlier, the initial time and final time for temporal pruning. This class in its turn will then do the pruning. After this process the view hash will be generated and signed using BUNGEE private key. Lastly, a JSON file will be generated and returned to the user. An example of a view can be seen in the attachments A.

In order to sign the view, asymmetric RSA 2048 keys were generated, where a private key is used to sign the views and a public key is used to verify the signature and thus validate its integrity.

5.2.3 Smart Contract

With regard to the development of smart contracts, two were developed. One for defining the asset and another with functions that perform the operations between BUNGEE and the blockchain, namely actions used during the snapshot generation, these are: *getAllAssets()*, *getAllAssetsKey()*, *getAllTxByKey()*. These have been explained in 5.2.1.

Several choices were made in terms of implementation and architecture of BUNGEE, these being the development of the code in TypeScript in order to meet the plugins previously developed on Hyperledger Cactus, Another decision was to develop the code using object-oriented programming paradigm to achieve modularity for easier troubleshooting and effective problem-solving.

We also chose to use Hyperledger cactus in order to have access to its ledger connector which allows us to connect with the selected blockchain, Hyperledger Fabric. The following figure 5.3 represents the Hyperledger cactus architecture presented earlier in chapter 4, however adapted to represent the connection between BUNGEE (client) and the blockchain (Hyperledger fabric).

A further choice was to use Hyperledger Fabric as a blockchain since it allows components, such as consensus and membership services, to be plug-and-play. Its modular and versatile design satisfies a broad range of industry use cases. It offers a unique approach to consensus that enables performance at scale while preserving privacy. The figure 5.4 represents a high level architecture between Hyperledger Cactus plugin, Bungee and Hyperledger Fabric Blockchain, this is useful since it gives us a visual representation of all the components. Briefly, since both architectures have already been explained in previous chapters, BUNGEE is designed to use the Cactus ledger connector in order to connect to the blockchain and thus perform actions.

Given that BUNGEE was developed in a modular way, it is fairly simple to change the use case we want to analyze. To adapt it to the new use case it is only necessary to change two classes, State-Bin where we modify the variable type *values* to the data structure that best suits the scenario, for example changing from a list of numbers to a list of dictionaries where each dictionary can hold as many fields as we need. The second class is Plugin-bungee, changing the function *generateLedger-*

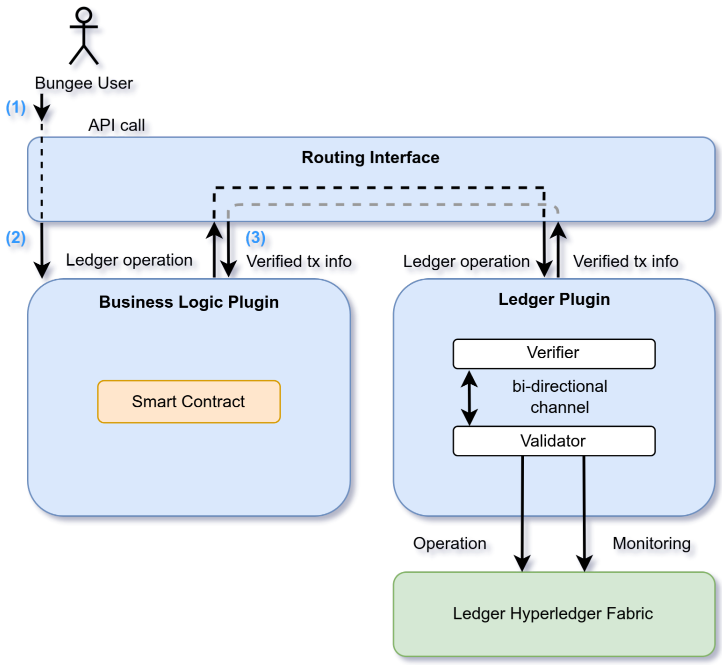


Figure 5.3: BUNGEE With Hyperledger Cactus Architecture

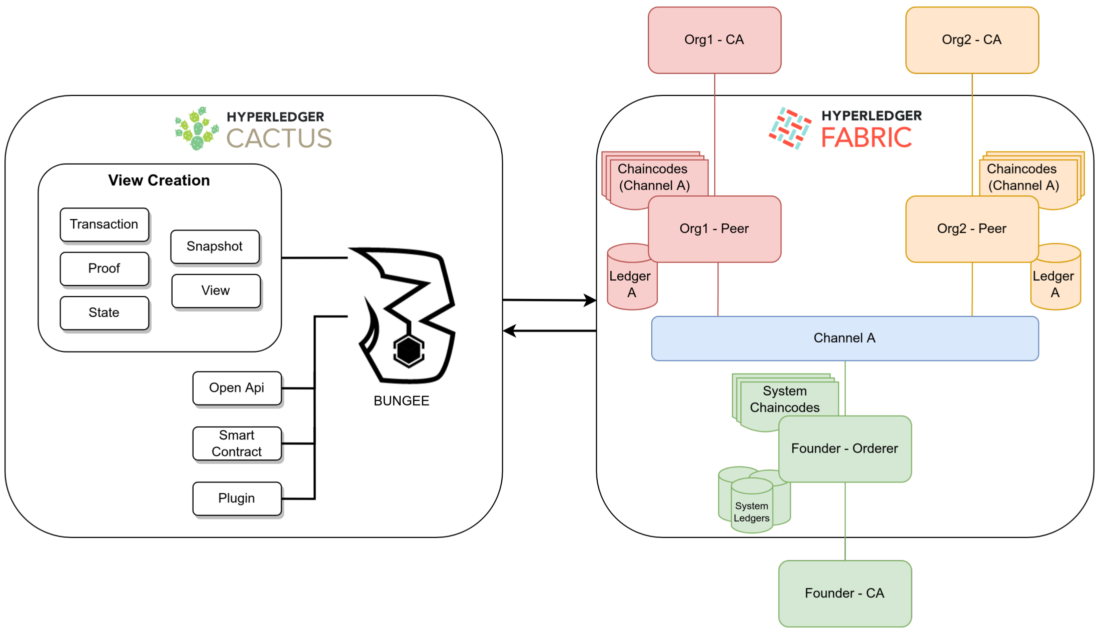


Figure 5.4: Global BUNGEE Architecture

States() in the section where it parses the values associated with an asset to match the data we actually need.

6 AuditView

Contents

After we have developed BUNGEE, we decided that it would be interesting to have the possibility to analyze the views without the necessity of inspecting the JSON file. Therefore, we created a web app in order to make it easier and more user-friendly to analyze the View. Before we developed the software, several functional and non-functional requirements were identified:

1. ***Data Integrity***, it should provide a service to detect unauthorized modification of data.
2. ***Usability***, the tool should be easy to operate by people with advanced computer knowledge, but also beginners.
3. ***Identifiability***, it should be possible to identify who audited the view.
4. ***Non-repudiation***, it should have a property that prevents an entity from denying previous commitments or actions.
5. ***Date Verifiability***, it should provide the possibility to verify creation date.

Once we become aware of the existing needs, it was clear what we should include in the output file. The following figure 6 represents the information that a user will get after performing an audit through AuditView. In this example the signature and hash fields have been reduced in order to simplify their representation.

Let's start by exploring data integrity, given that if data is changed intentionally or unintentionally, the audit loses its value, and is no longer valid. In order to ensure the data integrity, each auditor has access to a set of asymmetric RSA 2048 bit keys, generated at the same time as the auditor account is created. These pair of keys allow each of them to sign the audit with their private key, thus taking advantage of the properties offered by the asymmetric keys. In order to create the signature for the audit file, the data is first hashed using SHA-256, then the signature is created (*Auditor Signature, line 18*) and later included in the file.

Subsequently to the creation of an audit, it is possible to validate its integrity. To do this, we will only need the author's public key and the audit file. In order to do this validation, AuditView reconstructs the audit to memory, generates the corresponding hash and verifies the signature using the public key. Once AuditView have the result, it will show to the user if the audit still valid or has been tampered with. Regarding the audit view, we identify the assets (line 8-12) that were audited, since AuditView gives the possibility to select specific assets without having to analyze all of them. Further to the View, we also present the id, hash and signature (lines 13, 14, 15 respectively).

Concerning the auditing part, we have some pertinent information, such as the date and time of the audit creation (line 3,4 respectively). There are also the auditor's information, name and email (line 5, 6 respectively). This information can be changed in the future and more information can be included, such as the organization the auditor belongs to, phone number, among others. One feature of AuditView is to verify the View integrity, therefore, before the audit is performed, the auditor is informed if the view is valid or if it was tampered with. In both cases we offer the possibility to perform the audit, however in the final file we display if the view is considered valid (line 16).

```
1  {
2    "Audit": {
3      "Date": "16-10-2022",
4      "Time": "08:31:50",
5      "Auditor Name": "afonsomarques",
6      "Email": "afonso.m.marques@tecnico.ulisboa.com",
7      "Comments": "Comment about view audit",
8      "Assets Analyzed": [
9        "ASSET1",
10       "ASSET2",
11       "ASSET3"
12     ],
13     "View Id": "efa36dde-1d6c-4307-885f-72ce06bc8a73",
14     "View Hash": "1028fd480e003b3357c56306e66ac06fb793f444175da5164",
15     "View Signature": "QLldhIAjoc6Ku4yKildkF0SWG2q3PXezr6l53K6Od6QN",
16     "Is View valid": true
17   },
18   "Auditor Signature": "FgK1nhrwFs8HwTxpWftnhxrSbIMioe3mcNXzG6ZVLWD",
19 }
```

Listing 6.1: AuditView Output Example

Having introduced AuditView's functionality, we will now briefly present the layout developed for each of the web pages[1]. Since auditing requires information about the auditor, the auditor must be registered 6.1, for this he has to provide his name, email, password and password confirmation. In this form it is validated if the email already exists, if it is a valid email and if the password is considered secure. To be considered secure it must satisfy four criteria: it must be longer than six characters, it must not correspond to the email address, it must not correspond to the name and it cannot be a common password. After all fields are filled in, the auditor account is created and its asymmetric keys are generated.

[1] In appendix A you can find the complete images of the following figures: 6.1, 6.2, 6.3.

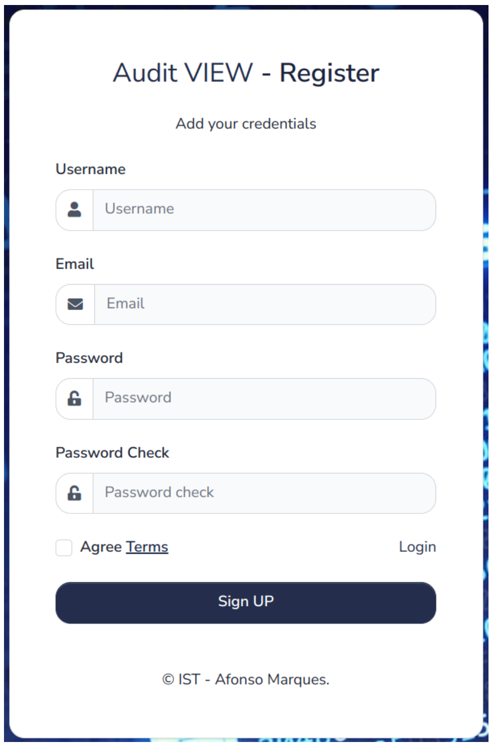


Figure 6.1: Register Page

In case the auditor's account already exists, he only needs to enter his email address and password 6.2.

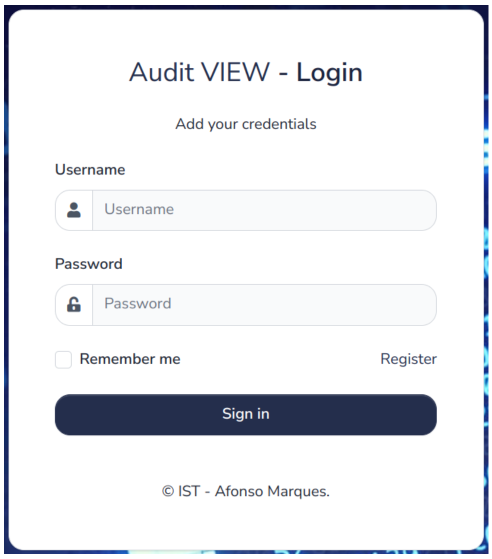


Figure 6.2: Login Page

On the main page the user will have the possibility to choose between two main features, option one, View auditing or option two, to check audit file integrity. The user will also have this choice through the navigation bar present on every page of AuditView 6.3.

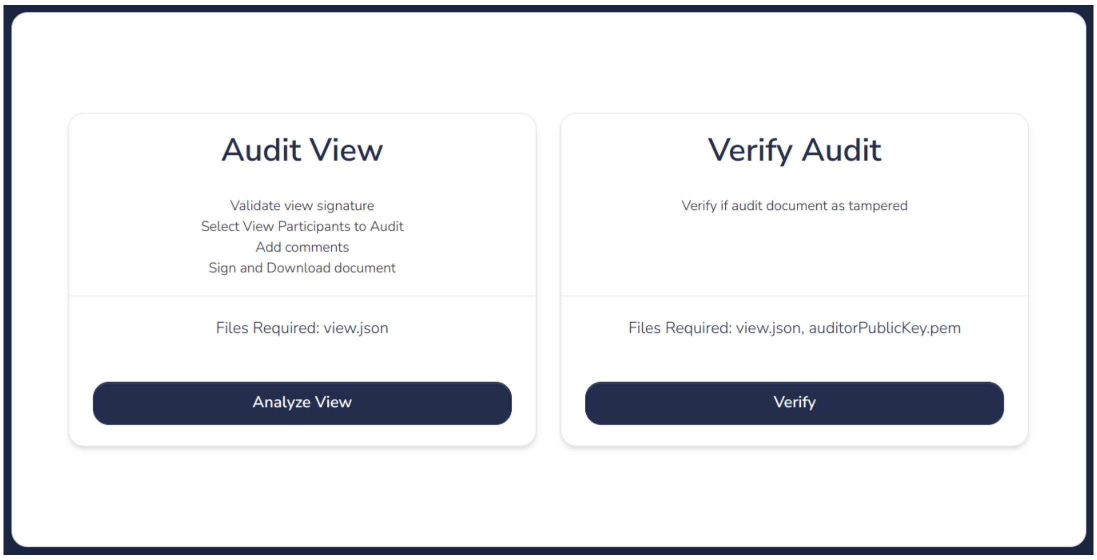

Audit View
Validate view signature
Select View Participants to Audit
Add comments
Sign and Download document
Files Required: view.json
Analyze View
Verify Audit
Verify if audit document as tampered
Files Required: view.json, auditorPublicKey.pem
Verify

Figure 6.3: Home Page

In case option one, view auditing, is chosen, the user will have to upload the view, however this view must have the expected format as presented in figure A. In case the view is not in the correct/expected format, a message will be displayed to the user so that he can upload a view with the correct format 6.4.

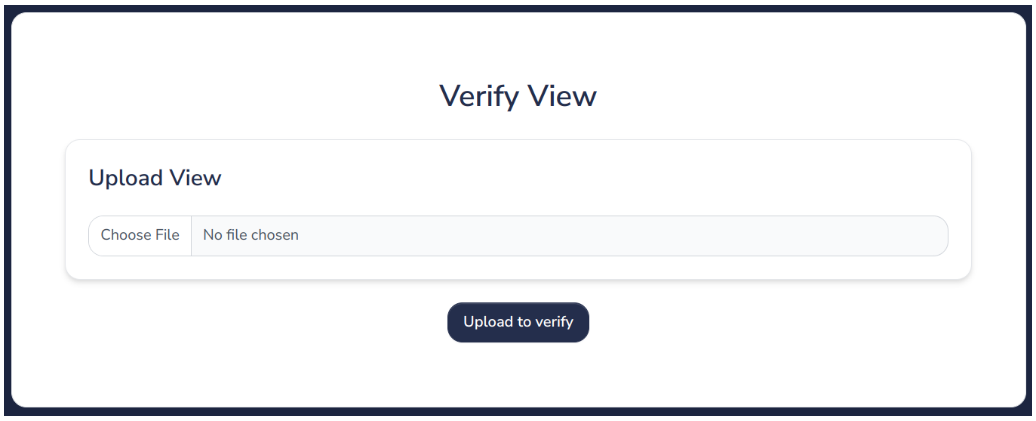
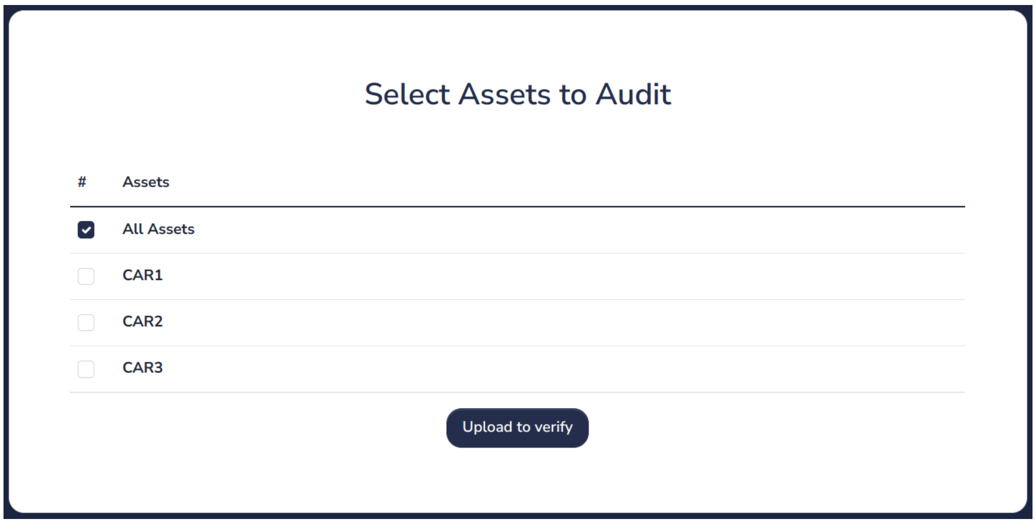


Figure 6.4: Upload View Page

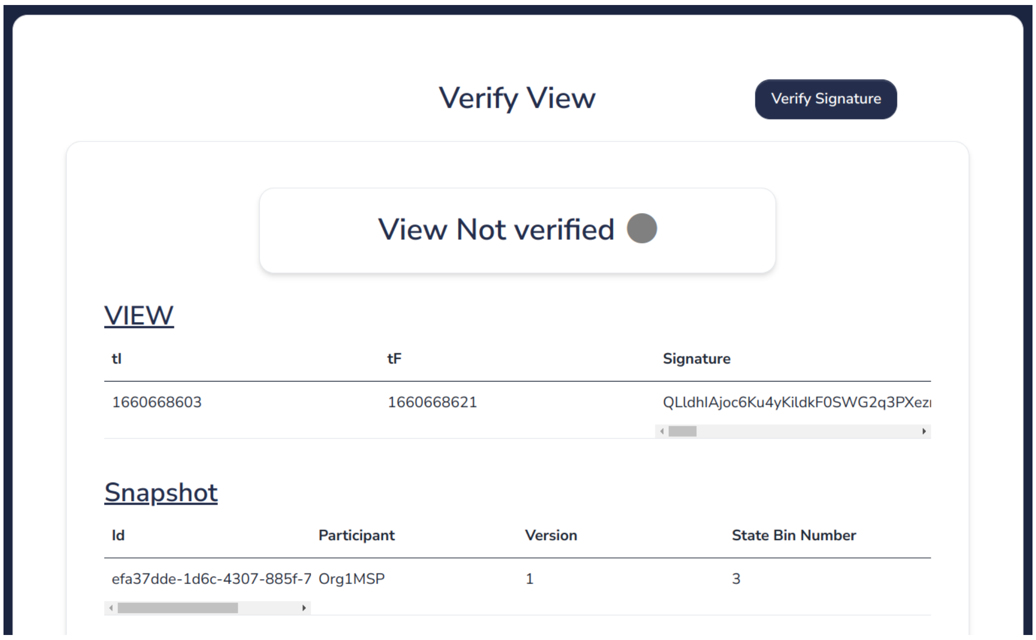
Once a view has been uploaded with the valid format, AuditView will rebuild the View into memory in order to be able to present all the assets contained in the view to the user. He then can select specific assets or perform a complete audit of all assets 6.5.



#	Assets
✓	All Assets
☐	CAR1
☐	CAR2
☐	CAR3

Figure 6.5: Select Asset Page

After the assets have been chosen for analysis, the auditor can do a brief overview in order to understand if he has chosen all the assets he intends to analyze, however, at this moment the signature has not been validated yet, therefore it is not possible to know if the View is valid. This page is useful since it allows the user to reconfirm the selected assets. To verify if the View is valid the user has to press verify signature 6.6.



VIEW

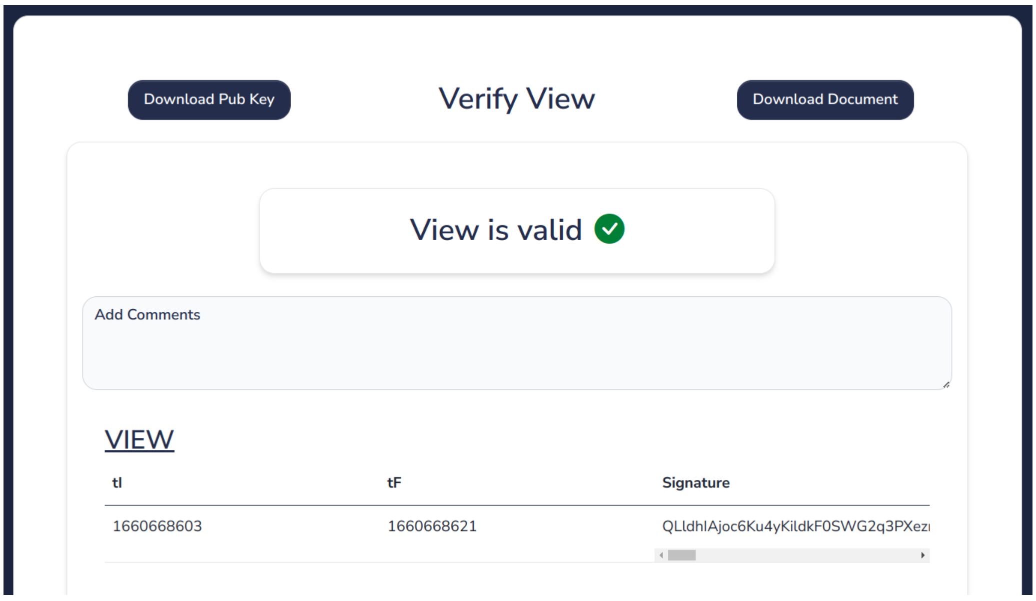
tI	tF	Signature
1660668603	1660668621	QLldhIAjoc6Ku4yKildkF0SWG2q3PXezı

Snapshot

Id	Participant	Version	State Bin Number
efa37dde-1d6c-4307-885f-7	Org1MSP	1	3

Figure 6.6: Preview View Page

After the user has pressed verify signature he will be redirected to the following page 6.7 where he will get confirmation about the View integrity and then proceed to the auditing page. On this page he can also download the audit containing the fields presented in 6, the auditor also as the possibility to download his public key that will be used to verify the audit integrity in the future by himself or another auditor. The name chosen for the audit file corresponds to the View Id that was analyzed.



VIEW

tI	tF	Signature
1660668603	1660668621	QLldhIAjoc6Ku4yKildkF0SWG2q3PXezı

Figure 6.7: Valid View Page

As option two, we have verify audit 6.8. Similar to the view validation, the user will have to upload a file, however, in this case it will be the audit file. It will also have to upload the public key associated with the auditor.

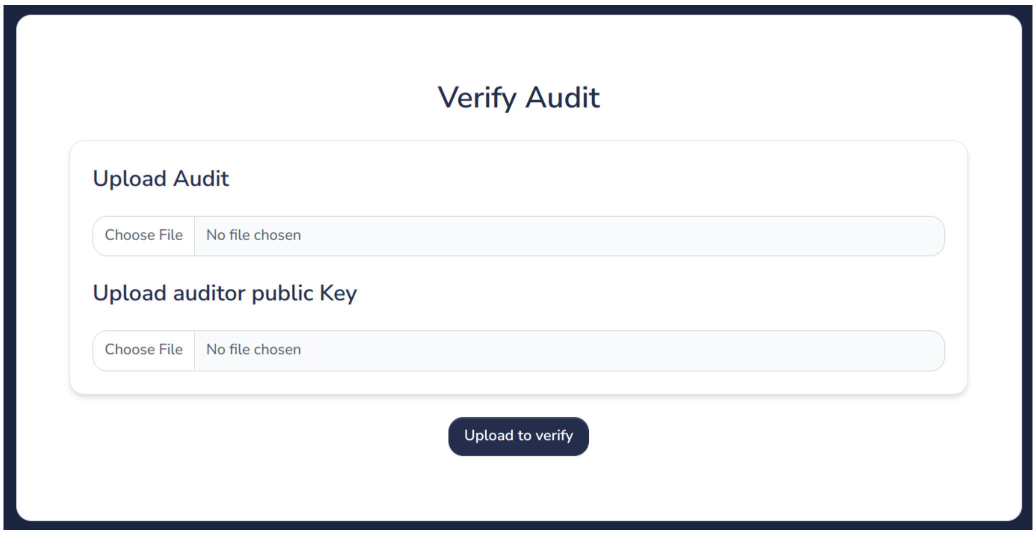


Figure 6.8: Upload Audit Page

To finalize, the user receives feedback regarding the audit integrity. In the figure 6.9, we have an example of the feedback corresponding to an audit that was tampered with.

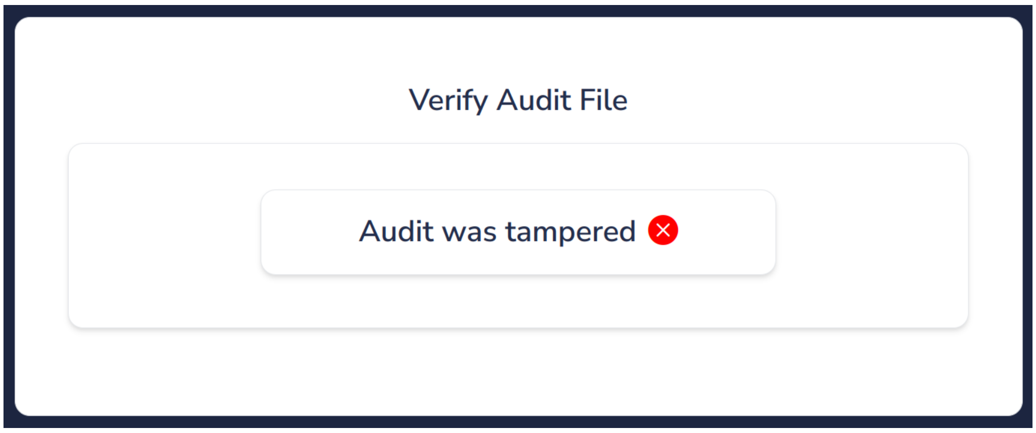


Figure 6.9: Audit Validation Page

6.1 Implementation Choices

In order to develop the AuditView web app we have used Django. Django is a high-level Python web framework that enables rapid development of secure and maintainable websites, it is also free and open source with great documentation. We choose this framework because of the following characteristics:

- ***Complete***, Django provides almost everything developers might want to do "out of the box".
- ***Versatile***, Django can be used to build almost any type of website
- ***Secure***, Django helps developers avoid many common security mistakes by providing a framework that has been engineered to protect the website automatically. It enables protection against many vulnerabilities by default, including SQL injection, cross-site scripting, cross-site request forgery and clickjacking.
- ***Scalable***, Django uses a component-based "shared-nothing" architecture (each part of the architecture is independent of the others, and can hence be replaced or changed if needed).
- ***Maintainable***, Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In our case *Model View Controller* (MVC) pattern.
- ***Portable***, Django is written in Python, which runs on many platforms.

When it comes to how AuditView handles user requests it has been divided into four main parts, these are: urls, view[2], template and model as we can see in the figure 6.10.

First the user starts by doing a HTTP request corresponding to a given URL *(1)*, then this request is forwarded to the corresponding view *(2)* which in turn requests the proper template *(3)* and reads and/or writes data from the model *(4)*. After the requests have been placed, the template will be returned *(5)* in order to present the data to the user *(6)* and thus inserted into the view. Lastly an HTTP response will be returned to the user*(7)*. Since we are using the MVC pattern, our *Model* is a central component of the pattern which manages the data, logic and rules of the application, *View* is used to represent the information and *Controller* accepts user input and converts it to commands for the model or view.

[2]It is important to note that in this case the word view does not refer to the Blockchain View but to what is presented to the user by the web app.

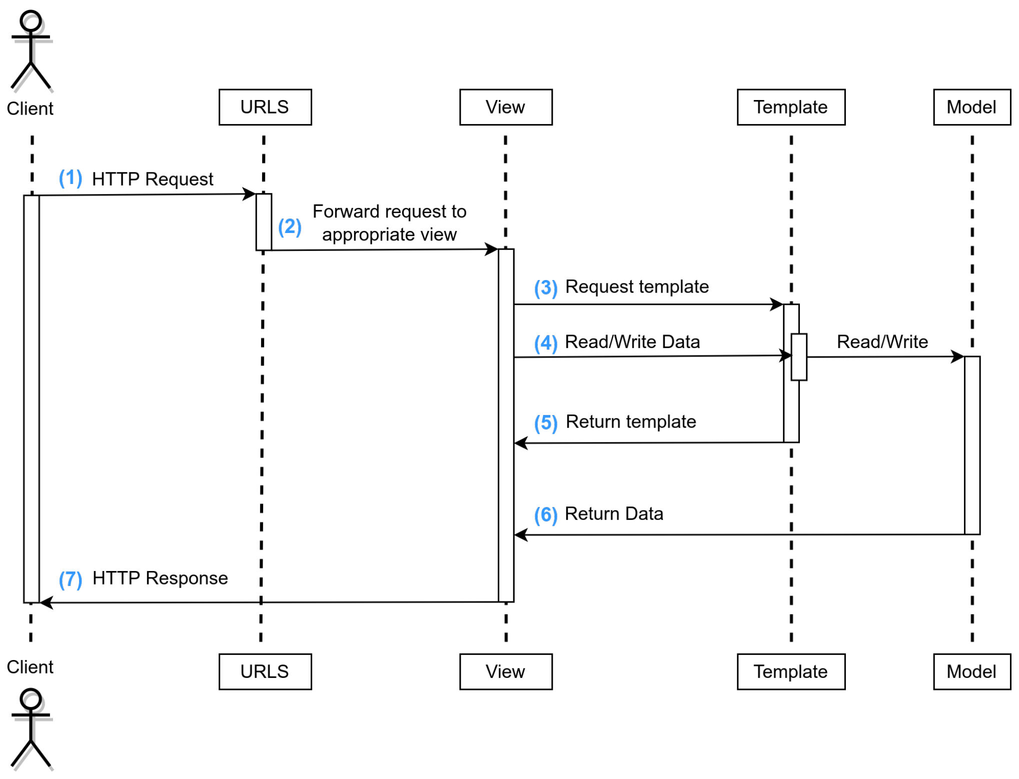


Figure 6.10: General Request Diagram

6.1.1 Signature Validation

To understand how the validation of the View generated by BUNGEE and validated by AudiView is done, we will look at the following figure 6.11.

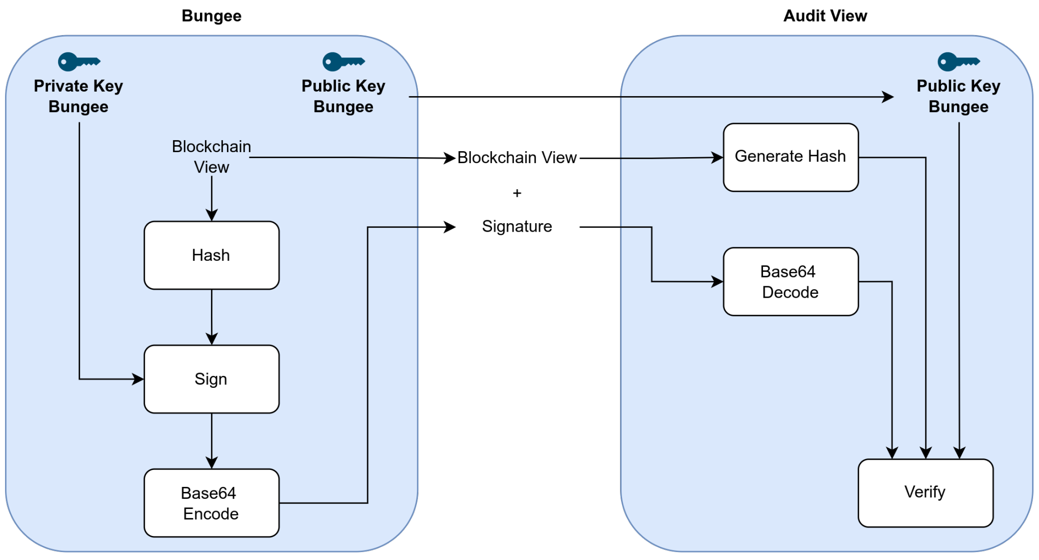


Figure 6.11: Signature Validation

Before we discuss the process of validation it is important to understand how the view is signed. As explained in section 5 after creation of the View is generated the hash using SHA-256, a cryptographic hash function, then using the library *crypto* of typescript is made the signature using the private key of BUNGEE, however this signature is in byte array format having the necessity to encode to base 64 so we can transmit the data without any losses in JSON format. After the view is finalized, it can be analyzed by the AuditView tool. To ensure the View integrity, the signature must be valid, and this only happens if no changes have been made since its creation. To validate the view, BUNGEE will share its public key with AuditView, this action only needs to be performed once since, AuditView will store the key. After this step, we can then upload the view, where AuditView will rebuild the data into memory.By having the data in memory it will do a similar process to BUNGEE in order to generate the hash of the view, and the reverse process to the signature, thereby decoding the base 64 to an array of bytes allowing the signature to be validated. With all the elements present, the view integrity is validated, using the decoded signature, the hash and BUNGEE public key. If the data has been tampered with, this will be the moment when we have the confirmation, since the function created for validation will return a warning to the user.

7

Evaluation

Contents

This section will present the tests performed on each developed tool and the outcame of these tests. In certain cases it is common to compare the developed solution with previously existing solutions, however, in our case, since we are working at the frontier of knowledge, it is not possible to take advantage of this comparison, since there are no tools to compare yet, thus making the evaluation even more challenging.

7.1 BUNGEE Evaluation

In order to evaluate BUNGEE, two types of tests were conducted, namely unit tests and performance tests. It is important to highlight that the obtained results were influenced by the performance of the machine used, so if the same tests were executed on a different machine, the values obtained would be slightly different.

The machine used as the following specs: Memory Ram 16Gb, Processor Intel® Core™ i7-7700HQ CPU @ 2.80GHz $\times$ 8, Graphics Card GeForce GTX 1050 Mobile.

For a tangible data set, 30 tests were performed for each number of transactions (10, 100, 500). From these thirty tests, ten of them were discarded in order to eliminate the outliers and have more consistent data. This step is particularly important since outliers are problematic. They represent measurement errors and/or processing errors and therefore impairing the consistency of the data.

Initially, we started by using the Hyperledger Caliper tool, which is a blockchain benchmark tool. Which allows users to measure the performance of a blockchain implementation with a set of predefined use cases. Hyperledger Caliper produce reports containing a number of performance indicators to serve as a reference, having the capability to analyze the Hyperledger Fabric blockchain. However, we soon realized that it was not feasible since the integration with Hyperledger Cactus is still very limited, so we had to find another strategy to solve this challenge.

To overcome this challenge we decided to measure how long the most important functions took to perform. For this purpose timestamps were placed throughout the code in order to record the initial time when the function started and the final time when it ended. After we had all these values, it was only necessary to compute the elapsed time, subtracting from the final time the initial time.

At the end of all tests being executed, the outliers were then removed. To this end, we start by calculating the first and third quartile (*Q1* and *Q3*). Then evaluate the interquartile range, which involves the subtraction between *Q3* and *Q1*, *IQR = Q3 - Q1*. Once we have the IQR, we calculate the lower bound (*Q1 – 1.5*IQR*) and the upper bound (*Q3 + 1.5*IQR*). Lastly we remove all the values below the lower bound and above the upper bound, thus obtaining a more homogeneous dataset of values.

In the sequence we will introduce three figures that represent the times required to perform the most important functions of BUNGEE. It is important to take the scale into consideration when these are analyzed, since they are all different. This decision was made since the execution times of the functions have significantly different ranges of values, therefore it was necessary to change the scale to have a better representation.

The figure 7.1 represents the chart of time versus number of transactions that each of the functions interacting with the blockchain requires. In order to make the values more perceptible, they were placed in the table 7.1.

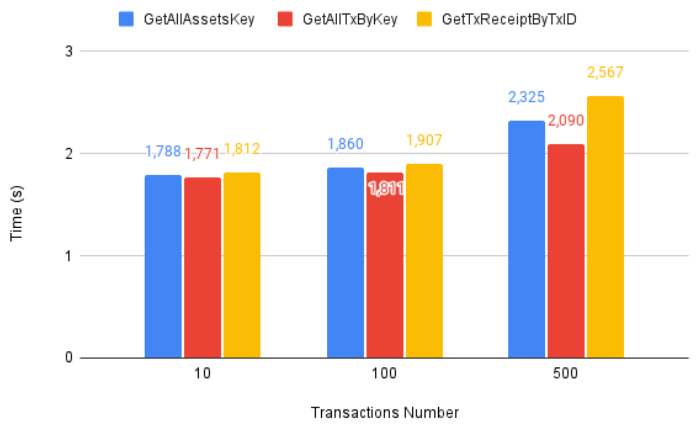


Figure 7.1: Number of seconds used by the functions: GetAllAssetsKey, GetAllTxByKey, GetTxReceiptByTxID

Tx Number	GetAllAssetsKey	GetAllTxByKey	GetTxReceiptByTxID
10	1,788347s	1,771108s	1,812183s
100	1,860473s	1,811004s	1,906563s
500	2,325358s	2,090279s	2,567222s

Table 7.1: GetAllAssetsKey, GetAllTxByKey, GetTxReceiptByTxID Values

The following figure portrays the time required by the *GenerateLedgerStates* function to process the number of transactions selected. As we can see, the value is higher than the functions shown in the figure 7.2, this happens because the function *GenerateLedgerStates* invokes the three previous functions. This means that the time it takes is the sum of the previous times, plus the processing time of the remaining function.

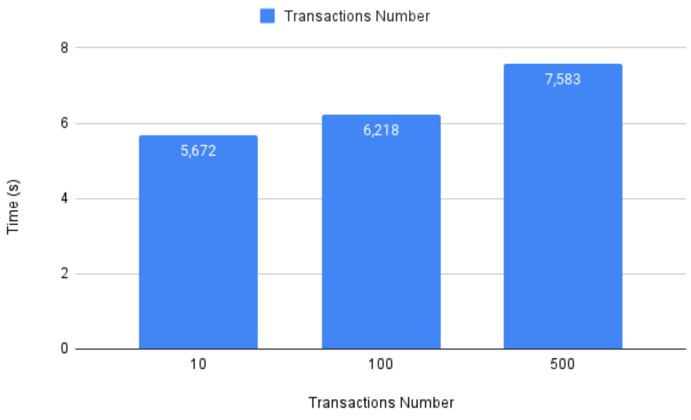


Figure 7.2: Number of seconds used by the function: GenerateLedgerStates

Tx Number	GenerateLedgerStates
10	5,671638s
100	6,218040s
500	7,582860s

Table 7.2: GenerateLedgerStates

Lastly, we have the figure 7.3, which, as the previous ones, represents the time spent by the functions *GenerateSnapshot* and *GenerateView* in detriment of the number of transactions.

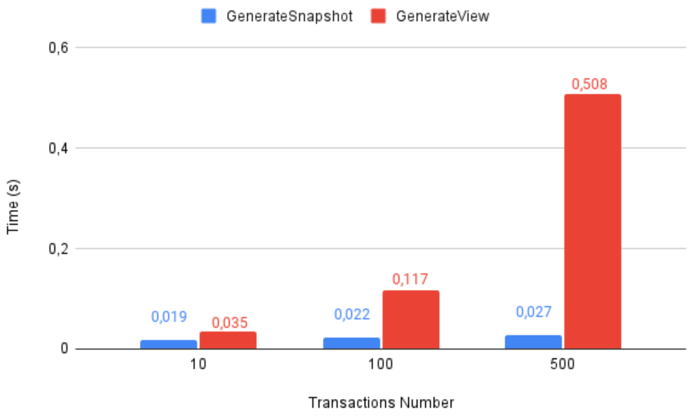


Figure 7.3: Number of seconds used by the functions: GenerateSnapshot, GenerateView

Tx Number	GenerateSnapshot	GenerateView
10	0,018515s	0,035067s
100	0,021907s	0,116664s
500	0,027118s	0,507847s

Table 7.3: GenerateSnapshot, GenerateView Values

Regarding the size of the views a file with 10 transactions will be about 21.1 kB, with 100 transactions it will be 202.8 kB and with 500 transactions it will be about 1Mb.

Our initial goal was to use a real use-case, however, since it was not possible to obtain a real use-case from a company that uses the technology, due to the sensitivity of the data, we created a use-case as close to reality as possible. This consisted of exchanging assets between peers. These had the possibility to sell and purchase assets. This way, it was possible to have data about asset owner's exchange inside our View. That being said, the values presented above not only vary with the change of the computer used to run the tests, but also vary with the change of the use-case. The values presented do not include the test ledger creation time nor the insertion of transactions, since in a production context these actions would not be performed by BUNGEE but by the peers in the blockchain.

It is important to note that BUNGEE was developed to meet the definition provided in the paper [19], which was submitted to the Journal of Parallel and Distributed Computing.During the development, some limitations were found and, consequently, solutions to overcome them were provided, thus helping to improve the paper.

7.2 AuditView Evaluation

As a means of testing AuditView we have used three types of methods, these were testing using low fidelity prototype[1], user testing with the tool completely developed and unit testing.

To perform the tests with the users, we started by understanding who would be a potential user of the tool, therefore we recruited six people. During recruitment, it was explained what the tests would be comprised of. The recruited people are computer engineering students, their ages range between twenty-one and twenty-three years old, and they are proficient when it comes to computer skills. The tests were divided into two sessions, the first session consisted of: Presentation of the tool, its purpose, and interaction tests using a low fidelity prototype. The second session involved testing the fully developed tool. For both sessions, a script of directives was made so that the users would know what was required and thus perform the actions.

[1] Low-fidelity (lo-fi) prototyping is a quick and easy way to translate high-level design concepts into tangible and testable artifacts. The first and most important role of lo-fi prototypes is to check and test functionality rather than the visual appearance of the product.

In the first session we started by presenting why the tool was developed, then we presented the functionalities we intended to develop, and finally we did the usability test using the low fidelity prototype. This prototype was developed in *Figma*[2] which is an interface design tool. This test showed that after registration it was important to give feedback to the user in order them to verify that the actions had been carried out successfully and not redirect directly to the main page, we realized that we should change the way we presented the two main functions to be more intuitive to use, and in the end we realized that it was important to have a navigation bar to facilitate navigation through the tool.

The second user test took place some time later, when the tool was already fully developed and the previous feedback incorporated. We could conclude that there was no difficulty in carrying out the requests made in the script for the users. Therefore, we can consider that the first session was very important once it made the tool easier and more intuitive to use, and this was confirmed in the second session.

These tests introduced some degree of complexity since it was necessary to take into account the availability of these six people, the development of the functionalities, and the stipulated deadlines.

Finally, we have the unit tests, these were the simplest to do since there was no need to manage logistics as in the previous tests. In order to perform the tests, we have used the *unittest* framework, which was originally inspired by JUnit. This tool gave us the possibility to make test cases, thus verifying the specific response to a particular set of inputs. Besides the unit tests, suit tests were also performed, consisting of a collection of test cases executed together. This way, all the functionalities presented were tested, and some problems were detected and subsequently rectified. These unit tests were performed between the first and second session with the users, therefore allowing us to present a stable version of our tool.

[2]https://www.figma.com/

8 Future Work

As we can see throughout this document, this was a very extensive work both in terms of features and technological stack, however there are still several aspects that would be very interesting to see developed with regard to this theme.

One of the functionalities to be developed in BUNGEE would be the View Merging part, which would be used when there are two views that we want to see in a single (unified) view. This functionality could be invoked on the fly after a view has been generated, or even when two previously generated views already existed. The following figure 8.1 incorporates the previously shown diagram 5.2, however with the details for a future implementation, thus including View Merge.

With the code that is already developed it won't be hard to develop this functionality, the steps needed would be only six and two of these were optional. In the figure 8.1 these steps are represented between the numbers 11-16. In order to develop this function, BUNGEE would have to provide the ability to upload existing views and validate their integrity or the ability to retrieve them (11). After having the two views in memory, it would be necessary to reconstruct them so that they could be represented by the classes that constitute them, thus creating an extended state (12). The following step would be to merge the classes in order to have a unified view (13). After having the view unified, it would be necessary to do the signature (14) of the view, similarly to step (9) explained before. As optional steps, we would have the persistent publication in the blockchain (15) and its proof (16). At last, the unified view would be returned to the client.

It would also be interesting to test BUNGEE using other blockchains such as Ethereum. This change of blockchain would not introduce much resistance since all the logic is already developed. It would only be necessary to change the plugin and possibly rename some functions in the smart contract to comply with the nomenclature used in the Ethereum blockchain.

In the future, if we decide to develop something to complement BUNGEE, it would be interesting to have a tool that could validate each of the transactions returned in the snapshot section, and verify the transactions related to an asset and its endorsements.

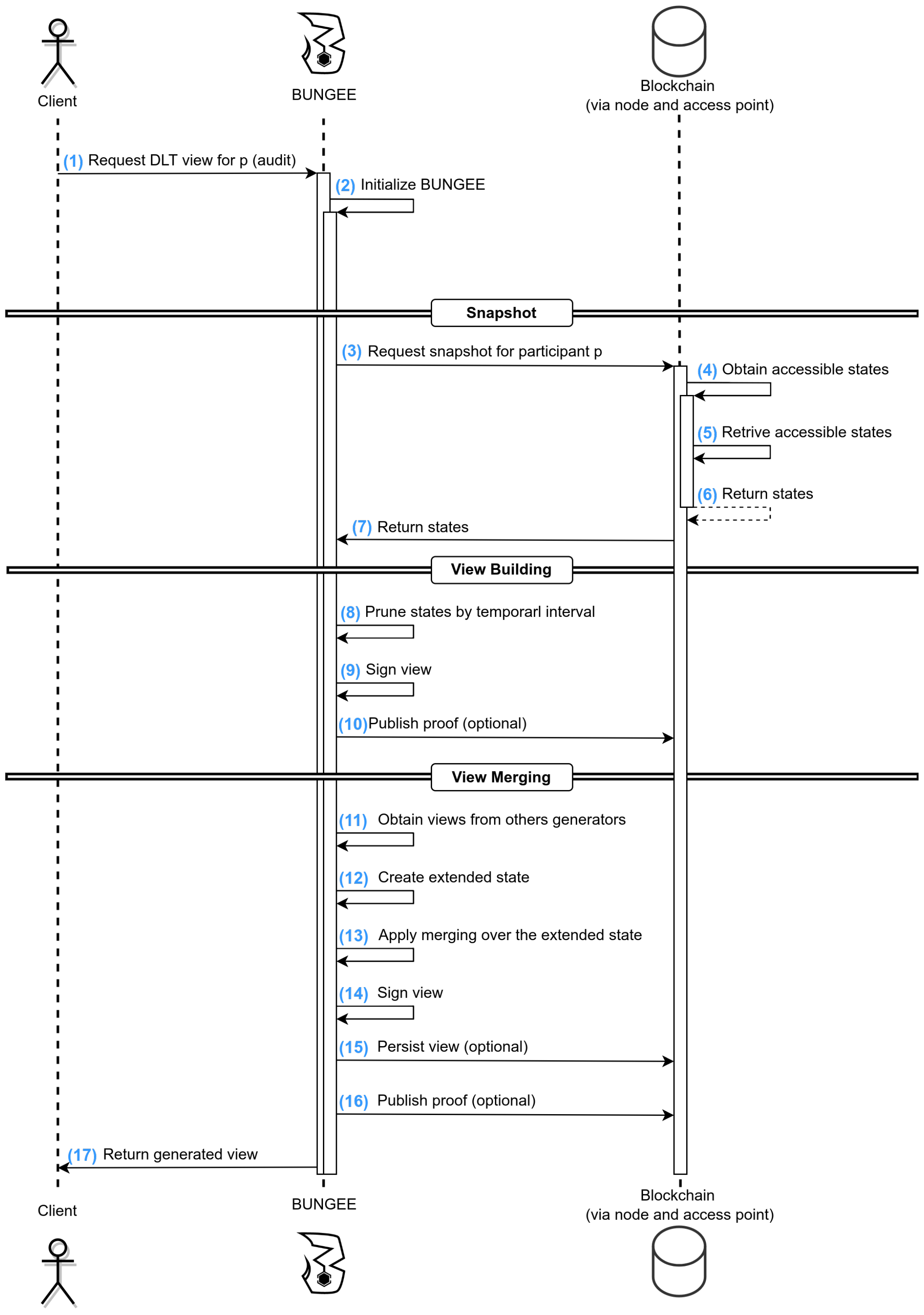

Client
BUNGEE
Blockchain (via node and access point)
(1) Request DLT view for p (audit)
(2) Initialize BUNGEE
Snapshot
(3) Request snapshot for participant p
(4) Obtain accessible states
(5) Retrive accessible states
(6) Return states
(7) Return states
View Building
(8) Prune states by temporarl interval
(9) Sign view
(10)Publish proof (optional)
View Merging
(11) Obtain views from others generators
(12) Create extended state
(13) Apply merging over the extended state
(14) Sign view
(15) Persist view (optional)
(16) Publish proof (optional)
(17) Return generated view
Client
BUNGEE
Blockchain (via node and access point)

Figure 8.1: BUNGEE Diagram With Merge Feature

9

Conclusion

Blockchain audit is a rapidly advancing research topic. Nevertheless, state-of-the-art tools are still limited and can only be used with cryptocurrencies. In this document, we describe how can we visualize data partitions within a blockchain and perform arbitrary operations on it, for instance generating blockchain views to conduct a blockchain audit.

We introduced state-of-the-art for fields like blockchain (private and public), auditing, blockchain auditing even though it does not have an extensive repository of articles and scientific papers.

We proposed a solution where we give the possibility to auditors, cybersecurity experts, and in general, developers to have audits facilitated because different data partitions can be analyzed from a specific angle. We also discussed what would be the advantages and disadvantages of our solution.

We believe that we have contributed to the state-of-the-art in blockchain audit and blockchain interoperability by generally develop a tool which allows users to use blockchain views and also the possibility for stakeholders to audit the blockchain in an easier, faster and more automatic way. In the future we expect to see auditors, cybersecurity experts, and in general, developers using the view concept to enhance audits, utilizing BUNGEE in this way. During development, we realized that there is a pronounced learning curve, especially for those who are not experienced with blockchain related technologies.

This is a much broader area than we might think, however, after settling this curve, tool development becomes very straightforward. Something important and very positive to highlight is the proximity of the maintainers of all the Hyperledger projects to the developers.

In conclusion, as we can see, this theme is undoubtedly very innovative, and can be applied to real cases, bringing great benefits that until now did not exist.

Bibliography

[1] P. J. Taylor, T. Dargahi, A. Dehghantanha, R. M. Parizi, and K.-K. R. Choo, “A systematic literature review of blockchain cyber security,” *Digital Communications and Networks*, vol. 6, no. 2, pp. 147–156, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864818301536

[2] D. S. Evans, “Economic aspects of bitcoin and other decentralized public-ledger currency platforms,” *University of Chicago Coase-Sandor Institute for Law & Economics Research Paper*, no. 685, 2014.

[3] R. Lai and D. LEE Kuo Chuen, “Chapter 7 - blockchain – from public to private,” in *Handbook of Blockchain, Digital Finance, and Inclusion, Volume 2*, D. Lee Kuo Chuen and R. Deng, Eds. Academic Press, 2018, pp. 145–177. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128122822000073

[4] J. J. Sikorski, J. Haughton, and M. Kraft, “Blockchain technology in the chemical industry: Machine-to-machine electricity market,” *Applied Energy*, vol. 195, pp. 234–246, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261917302672

[5] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” http://bitcoin.org/bitcoin.pdf,” 2008.

[6] E. F. Jesus, V. R. Chicarino, C. V. De Albuquerque, and A. A. d. A. Rocha, “A survey of how to use blockchain to secure internet of things and the stalker attack,” *Security and Communication Networks*, vol. 2018, 2018.

[7] N. Rathod and D. Motwani, “Security threats on blockchain and its countermeasures,” *Int. Res. J. Eng. Technol*, vol. 5, no. 11, pp. 1636–1642, 2018.

[8] S. Zhang and J.-H. Lee, “Double-spending with a sybil attack in the bitcoin decentralized network,” *IEEE Transactions on Industrial Informatics*, vol. 15, no. 10, pp. 5715–5722, 2019.

[9] T. Koens and E. Poll, “What blockchain alternative do you need?” in *Data Privacy Management, Cryptocurrencies and Blockchain Technology.* Springer, 2018, pp. 113–129.

[10] D. Yaga, P. Mell, N. Roby, and K. Scarfone, “Blockchain technology overview,” *arXiv preprint arXiv:1906.11078*, 2019.

[11] P. Vasin, “Blackcoin’s proof-of-stake protocol v2,” *URL: https://blackcoin. co/blackcoin-pos-protocol-v2-whitepaper. pdf*, vol. 71, 2014.

[12] S. Solat, P. Calvez, and F. Naït-Abdesselam, “Permissioned vs. permissionless blockchain: How and why there is only one right choice,” *Journal of Software*, vol. 16, pp. 95 – 106, 12 2020.

[13] J. Mullarkey, “Case for the structured audit,” 1984.

[14] M. A. Geiger and K. Raghunandan, “Bankruptcies, audit reports, and the reform act,” *Auditing: A Journal of Practice & Theory*, vol. 20, no. 1, pp. 187–195, 2001.

[15] A. Hylton, “A km initiative is unlikely to succeed without a knowledge audit,” *Knowledge Board.[Consulta: 15 febrero 2009]*, 2002.

[16] A. Baev, V. Levina, A. Reut, A. Svidler, I. Kharitonov, and V. Grigor’ev, “Blockchain technology in accounting and auditing,” *Accounting. Analysis. Auditing*, vol. 7, no. 1, pp. 69–79, 2020.

[17] A. Ahmad, M. Saad, and A. Mohaisen, “Secure and transparent audit logs with blockaudit,” *Journal of network and computer applications*, vol. 145, p. 102406, 2019.

[18] D. Bonyuet, “Overview and impact of blockchain on auditing,” *International Journal of Digital Accounting Research*, vol. 20, pp. 31–43, 2020.

[19] R. Belchior, L. Torres, J. Pfannschmid, A. Vasconcelos, and M. Correia, “Is my perspective better than yours? blockchain interoperability with views,” 2022.

A

Appendix

```
{
  "View": {
    "tI": "1660668603",
    "tF": "1660668621",
    "snapshot": {
      "id": "efa37dde-1d6c-4307-885f-72ce06bc8a73",
      "participant": "Org1MSP",
      "version": 1,
      "stateBins": [
        {
          "id": "ASSET1",
          "version": 2,
          "values": [
            "30"
          ],
          "transactions": [
            {
              "id": "bb42b1045937aea67aec35fb40d6031d7086e40617bbb9f20d295a368",
              "timeStamp": "1660668612",
              "proofs": [
                {
                  "mspid": "Org1MSP",
                  "endorserID": "--BEGIN CERTIFICATE--\nMIICqjCCAlCgAwIBAgIUdK",
                  "signature": "MEUCIQDCv37Dg8Kcw4vDuMOZw7QdJyLCkwnDo8KLw6fDq3"
                },
                {
                  "mspid": "Org2MSP",
                  "endorserID": "--BEGIN CERTIFICATE--\nMIICpjCCAkygAwIBAgIUUY",
                  "signature": "MEQCIDXCo1zDlXXCt0DCggLDoxfDvsO4wrLDrMOEGsO9w6"
                }
              ]
            }
          ]
        }
      ]
    }
  },
  "Signature": "QLldhIAjoc6Ku4yKildkF0SWG2q3PXezr6l53K6Od6QN6IcG3d"
}
```

Listing A.1: View Example

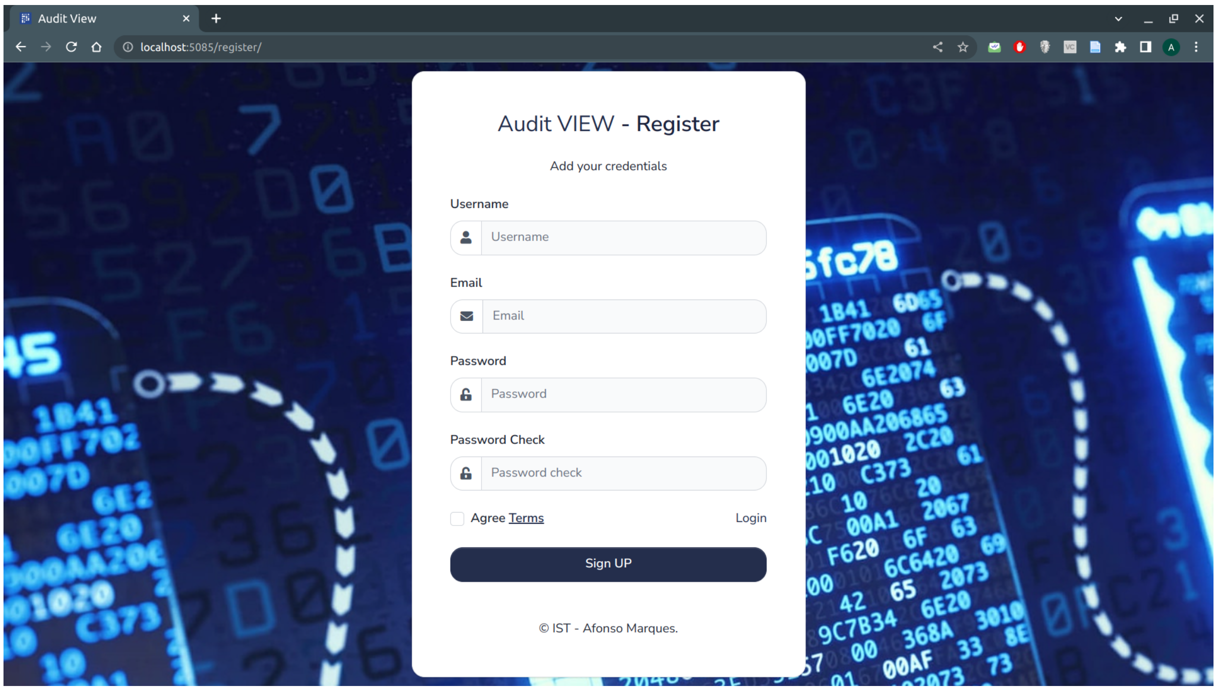


Figure A.1: Register Page Full Image

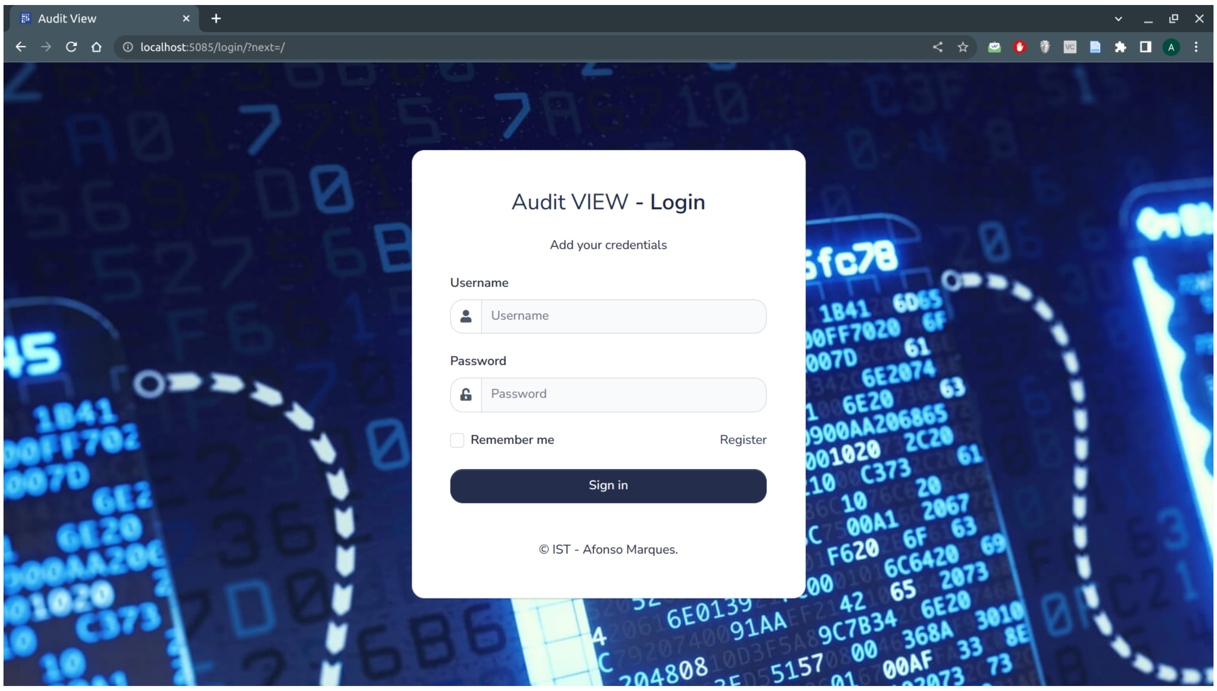
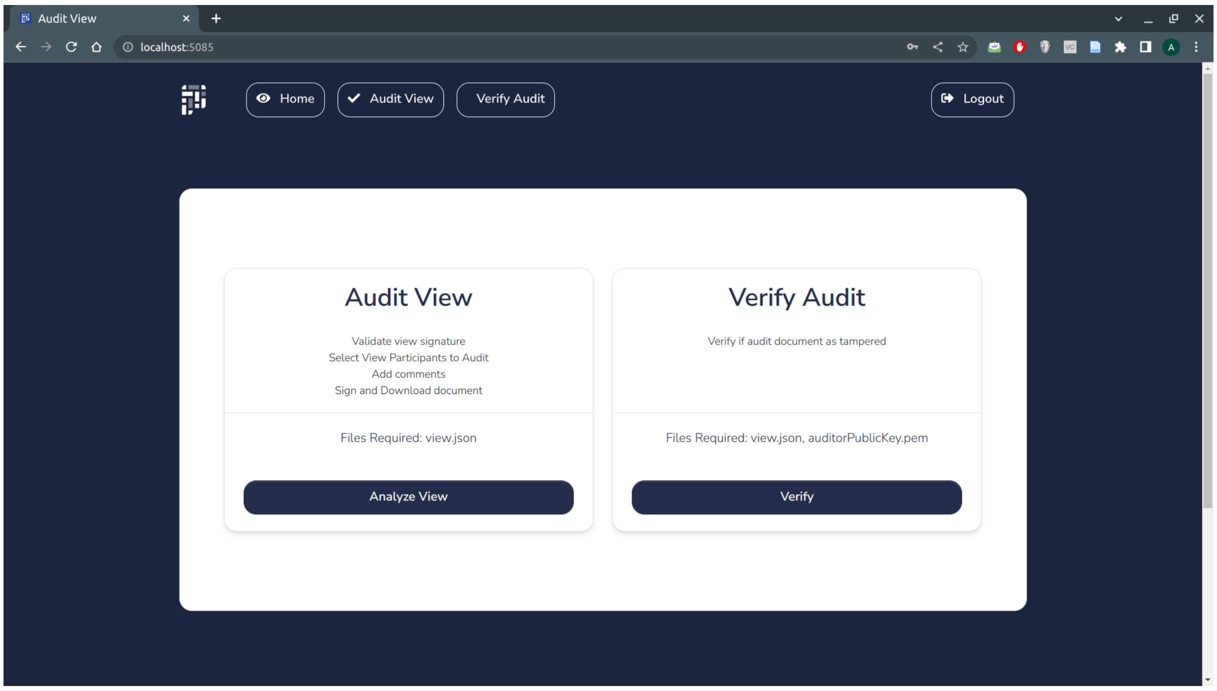


Figure A.2: Login Page Full Image



Audit View

Validate view signature
Select View Participants to Audit
Add comments
Sign and Download document

Files Required: view.json

Analyze View

Verify Audit

Verify if audit document as tampered

Files Required: view.json, auditorPublicKey.pem

Verify

Figure A.3: Home Page Full Image