University
of Stavanger

ASAHI CANTU
FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

NFT as a proof of Digital Ownership-reward system integrated to a Secure Distributed Computing Blockchain Framework

Master's Thesis - Computer Science - June 2022

I, **Asahi Cantu**, declare that this thesis titled, “NFT as a proof of Digital Ownership-reward system integrated to a Secure Distributed Computing Blockchain Framework” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master’s degree at the University of Stavanger.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

"Whereas most technologies tend to automate workers on the periphery doing menial tasks, blockchains automate away the center. Instead of putting the taxi driver out of a job, blockchain puts Uber out of a job and lets the taxi drivers work with the customer directly. "

– Vitalik Buterin, Co-Founder of Ethereum

Abstract

Today, the global economy is dependent on the Internet and computational resources. Although they are tightly interconnected, it is difficult to evaluate their degree of interdependence. Keeping up with the pace of technology can be a challenging task, mainly when updating the hardware and software infrastructure. Every day, corporations and governments are faced with this issue; most have been victims of cyber attacks, security breaches, and data leaks. The consequences are significant in monetary losses; damage remediation is unattainable, even impossible, in certain circumstances. The repercussions might include reputational damage, legal responsibility, and threats to national security (when attacks are carried out against critical infrastructures to control the resources of a country), to name a few. Similarly, data has become such an integral part of many industries that it is one of the most critical targets for attackers that often is encrypted by ransomware, stolen, or corrupted. Without data, many companies are not be able to continue operating as they do. The combination of all these factors complicates the ability of organizations to cooperate, trust, and share information in efforts to research and develop solutions for industry and government.

A promising technology can assist in significantly reducing the damage caused by the security threats outlined above: Blockchain technology has proven to be one of the most promising inventions of the twenty-first century for transmitting and protecting information while offering high reliability and availability, low exposure to attacks, protected encrypted data, and accessible to the entities willing to participate. Blockchain enabled the possibility to embed immutable data and compiled source code known as ‘smart contract’ where certain rules can be programmed to create business workflows.

This thesis report proposes a Blockchain-based infrastructure solution provided by ”Hyperledger Fabric” technology for companies to securely transmit and share information using the latest encryption and data storage technologies operating on the model of distributed systems and smart contracts. By presenting unique digital assets as Non-Fungible Tokens (NFT), the infrastructure is able to trust the integrity of the data, while protecting it from counterfeiting. Through the use of a Blockchain-based file storage system known as IPFS, and by connecting all the relevant elements together through a web-based application, it is possible to demonstrate that the implementation of such systems is feasible, highly scalable and a useful tool that many organizations can utilize to create new work systems and workflows for digital asset management.

Acknowledgements

I would like to thank my supervisor, Chunming Ron and Jiahui Geng for their guidance, contribution and help in the elaboration of this thesis project.

Family, friends and colleagues for their support and patience.

Finally, my gratitude to the University of Stavanger and the possibility to work on this topic.

Contents

0.1 Abbreviations

API	Application Program Interface	24
AWOL	Absence Without Official Leave	22
BFT	Byzantine Fault Tolerance	22
CA	Certificate Authority	29
CFT	Crash Fault Tolerant	27
CID	Content Identifier	31
DAC	Decentralized Autonomous Corporation	24
DAG	Directed Acyclic Graph	15
DAO	Decentralized Autonomous Organization	16
DAPP	Decentralized Application	21
DFS	Distributed File System	7
DLT	Distributed Ledger Technology	8
DMS	Data Management System	34
DSL	Domain-specific Language	27
EIP	Ethereum Improvement Proposals	15
ERC	Ethereum Request for Comments	8
EHR	Electronic Health Records	33
ETH	Ethereum Cryptocurrency token	52
IPFS	Interplanetary File System	7
IT	Information Technology	34
IoT	Internet of Things	35
KB	KiloByte	32

MPA	Multi-party authorization	35
NFT	Non Fungible Token	8
Nonce	Number only used once	23
OS	Operating System	38
PBFT	Practical Byzantine Fault Tolerance	22
PKI	Public Key Infrastructure	29
PoA	Proof of Authority	22
PoET	Proof of Elapsed Time	22
PoS	Proof of Stake	21
PoW	Proof of Work	13
P2P	Peer-to-Peer	30
REST	Representational State Transfer	41
TB	TeraByte	36
TCG	Trading Card Game	16
UAV	Unmanned Aerial Vehicle	34
UI	User Interface	41
USD	United States Dollar	36
WSL	Windows Subsystem for Linux	56

Chapter 1

Introduction

As Blockchain technologies attract the interest of worldwide industries for its intrinsic values, new possibilities unleash to promote mutual cooperation in benefit of building up and improving technologies securely and manageable. Companies can benefit from such systems when through inter-organizational trust, privacy and protection; improving and protecting businesses while cooperating and sharing information through a common system.

When it comes to large-volume data sharing and storage, it becomes vital to verify its authenticity and avoid plagiarism/counterfeiting. This thesis project proposes a system for the treatment of data as a digital asset through the concept of smart contracts to manage Non-fungibility in a Blockchain system, and a server/application infrastructure to expose the potential and possibilities that data ownership can have when it is brought into decentralized permissioned systems with Hyperledger Fabric as a Blockchain system and Interplanetary File System (IPFS) as a Distributed File System (DFS) system.

1.1 Background and Motivation

As digital era keeps evolving and cloud computing increases its power, users and companies are no longer in full control of their resources, rather than that, they led third party companies to store their information without fully knowing the way or locations it could be. In addition to this, recent cyber security risks and the progress of malware, ransomware and other harmful technologies have put the whole world into a cyber crisis. Ever since the creation of the interned such cyber attacks have been increasing exponentially and represent risks for the assets of the companies, countries and end users. Data is vital, data keeps breaching and leaking sometimes without event noticing but months after the damage has happened.

This brings the need for an implementation of a more secure way to store and manage data through the internet without suffering from the present security risks.

Decentralized systems and Blockchain technologies created in the last ten years offer a tremendous potential to bring organization into a new way to manage their virtual assets.

It is therefore the purpose of this thesis project to create an architecture and functional system to record unique ownership of digital assets in a permissioned blockchain (datasets ownership) while providing the infrastructure to allow its usage or deny it depending on the agreements of the system through the issuance of NFT's and smart contracts as an alternative solution that allows to alleviate the present cyber security risks while enabling data management securely and privately.

1.2 Objectives

The present work will demonstrate the functionality and potential of a digital asset management system via Blockchain and Non Fungible Token (NFT) smart contract with IPFS as a data storage repository.

- Create a permissioned system blockchain with the ability to store Digital Assets through the use and extension of the Ethereum Request for Comments (ERC)-721 protocol and expand its potential for file storage, data sharing and distribution.
- Use the protocol IPFS to store data and link its address in the blockchain
- Implement a consensus mechanism that incentivize cooperation and participation of different parties with the purpose to share, trust, compose and improve data structures and third party systems by implementing a Distributed Ledger Technology (DLT).
- Provide a baseline platform to allow the usage of NFT and promote mutual trust the system
- Expose alternative consensus mechanisms to extend the application of the system in other industries.
- Create a system able to represent ownership of certain digital assets through the emission of NFT

- Propose a system to transfer such NFTS between institutions as the equivalent of ownership transfer

- Link the ownership system with the identity and blockchain databases.

- Demonstrate that proposed functionality could enable institutions and corporations easy ways to cooperate and compute datasets by using ownership mechanisms

1.3 Approach and Contributions

This thesis work uses Hyperledger Fabric permissioned Blockchain framework as a base platform to implement an ERC-721 smart contract extension for the creation and certification of digital assets, which for the aim of this thesis is represented in data. It also implements an distributed data lake infrastructure through the usage of a private IPFS network, which allows storage of information in a reliable and trustless manner. When this data is linked to the ERC-721 smart contract, an NFT asset is created, and single entity properties like ownership, authenticity, transfer, royalties, etc, can be exposed to all its participants.

It demonstrate through the creation of a back-end and front-end example applications the potential of its usage through a simulated token minting and data sharing environment among different known parties where the unique source of trust is a central authority and the previous agreement of the rules (consensus protocol) through which the data can be generated, shared and transferred in the same manner as if a crypto asset was transferred. Since the environment has been implemented using Docker Containers Technology, its scalability and security are granted and can be easily implemented. The results can be visualized in the following github repository, the program can be downloaded and deployed in any computer system with specific requirements

- ERC-721 protocol creation and extension

- Consensus mechanism

- Infrastructure and Blockchain environment in Hyperledger Fabric

- REST API in the back-end for common collaboration with the blockchain

- Front new Web application as a simulation of work and data generation through the blockchain

- Creation of a private IPFS network and data persistence schema for common data sharing

A detailed analysis of the systems infrastructure and collaboration, the interpretation of roles and mechanisms that an organization will play to grant plain participation and collaboration over the blockchain systems in Hyperledger Data persistence and availability in the IPFS environment.

Generation of a proof of authenticity and non repudiation of data, consensus protocol, trough the exploration of the blockchain and closes application system.

1.4 Outline

The thesis projects consists of the following chapters:

1.4.1 Chapter 1. Introduction

- A brief explanation of decentralized systems and how they emerged with the invention of Bitcoin. Through the creation of smart contracts, the issuance of fungible and non-fungible tokens will be possible, both in a proto-version and in a more advanced form with the evolution of decentralized systems. Overview of NFT-based systems, their original applications, and their potential for industrial use
- Further explanation of permissioned blockchain systems and the role of Hyperledger Fabric as an open-source platform for industrial applications.
- A brief presentation of IPFS as a decentralized file system and its potential to use decentralized content-based storage solutions in private networks. These two systems: ”Hyperledger Fabric” and ”IPFS,” are merged to create an infrastructure able to mint NFT representing digital data assets with different capabilities able to solve modern industry data sharing and solve trust, authenticity, and security problems.

1.4.2 Chapter 2. Related Work

- Demonstration how how permissioned blockchain systems developed and deployed using ”Hyperledger Fabric” have helped industries expedite processes and open collaboration with other parties.

- Related work and applications of IPFS as a decentralized file system for the Web 3.0. Brief explanation of how this has helped to empower end users and how its application for the industry can create independence and full ownership of data with similar reliability as the provided by cloud companies.
- NFT systems based in the Ethereum and other blockchain networks.
- Similar approaches through the implementation of these systems in the public blockchain

1.4.3 Chapter 3. Approach

This chapter explains how ”Hyperledger Fabric” and ”IPFS” System were used together to incorporate a permissioned blockchain system to share data in a secure and self trusted manner.

- A demonstration of a custom smart contract developed specifically to hold the necessary instruction for NFT issuance.
- The concept of ”proof of ownership” and ”proof of authenticity” are incorporated as consensus mechanisms to avoid data counterfeiting and incentivize its curation, usability and reliability through multiple evaluation mechanisms.

1.4.4 Chapter 4. Experimental Evaluation

With all the elements provided in 3, a whole infrastructure is assembled and its functionality is demonstrated through the implementation of Linux containers as virtual servers emulation the whole systems. back-end and front-tend systems have been build in addition as this process allows to show the system in its full potential and propose further system escalation as required.

1.4.5 Chapter 5. Discussions

From proposed solution in 4 future work and other consensus approaches are discussed. Its scalability and evolution can be set in place for different industry applications beyond big data sharing.

1.4.6 Chapter 6. Conclusions

This chapter explains how after the development and incorporation of two unrelated systems it is possible to create reliable infrastructures with the potential to change the way companies currently use, store and share information inside and among other organizations.

Chapter 2

Related Work

For the elaboration of this thesis project sources with subjects related to blockchain technology and networking systems had to be consulted.

2.1 Blockchain

Bitcoin whitepaper, where it exposes for the first time ”A peer-to-peer version of electronic cash”Nakamoto [2008], by elimination the double spend problem in decentralized systems. It proposes an unbreakable system where any user with a computer can participate by just joining the network. It provides reliability and trustless system through the usage of asynchronous encryption, hashing algorithms, elliptic curves for random password generation, *Merkle trees*[1] and the concept of ”blocks” which are portions of data with fixed size recording the ledger. The system compensates computers who keep the system secure by solving a complex mathematical problem Proof of Work (PoW). Bitcoin brings security by democratizing access to all its participants without a central authority and with the generation of value-appreciation.

[1] Binary hash tree in which every ”leaf” (node) is labelled with the cryptographic hash of a data block, and every node that is not a leaf (called a branch, inner node, or *i*node) is labelled with the cryptographic hash of the labels of its child nodes. A hash tree allows efficient and secure verification of the contents of a large data structure.

2.2 Hyperledger Fabric

Hyperledger Fabric is an open source Framework supported by the Hyperledger foundation[2] created to implement permissioned distributed ledger platforms for industrial applications. It offers modularity and versatility for many enterprise projects requiring distributed trustless systems where many parties can interact by joining private channels where a ledger can execute smart contracts and adapt to the business rules without compromising their corporate business securityHyperlerdger [2022]. Unlike public Blockchains, *Hyperledger Fabric* uses different services to authorize access to the infrastructure through private channels where specific smart contract rules can be transacted to simply query the ledger or perform operations over stored data.

2.3 Distributed File systems and IPFS

DFS Is a file system spread across multiple locations over a Network. This allows programs to access or store isolated files the same way as they do local ones, allowing them to access files from any network or computer.

Using a Common File System, the DFS makes it possible for users of physically distributed systems to share data and resources. For example, using a Local Area Network (LAN) to connect workstations and mainframes is a Distributed File System. The DFS consists of two components:

- Transparency of location is achieved through the namespace component.
- File replication provides redundancy.

This combination of components can improve data availability in the event of failure and heavy load by allowing data sharing across different locations to be logically grouped under one folder, which is called the DFS root. Using both components of the DFS architecture is not required; one can use the namespace component without using the file replication component, and one can use the file replication component without using the namespace component between servers.

[2] *Hyperledger Foundation*Foundation [2022a] is a non profit organization that brings together all the necessary resources and infrastructure to ensure thriving and stable ecosystems around open source software blockchain projects.

2.3.1 The IPFS System

IPFS is a peer-to-peer distributed file system designed to connect all computing devices through the same file system. In some ways, IPFS is similar to the Web. However, IPFS could be viewed as a swarm of *BitTorrent*[3] peers exchanging objects within a single *Git*[4] repository. Most commonly when navigating on the Web, data is requested and retrieved by its location (better known as location-based address), meaning that no matter the type or morphology of the data, it will be retrieved by the address where it exists. This implies that data can be amended, exchanged and even counterfeited. In contrast, IPFS provides a content-addressed block storage model where instead of a hyperlink pointing to the location of an address, a content-addressed hyperlink ”knows” the content of the file that should be retrieved. The result is a *Generalized Merkle Tree Directed Acyclic Graph (DAG)*[5].Benet [2014]

The IPFS protocol combines a distributed hashtable, an incentive-based block exchange system, and a self-certifying namespace. In IPFS, there is no single point of failure, and nodes do not need to trust one another.

2.4 NFT-Related systems

As mentioned before, the Ethereum Foundation was first on applying the concept of Smart Contracts over the Blockchain. When this happened a set of application-level standards, name registries, library/package formats, programming language and all the structure of the framework were officially documented in a repository known as ERC in 2019. In this site many guidelines known as Ethereum Improvement Proposals (EIP) can be found on how to create code for different smart contract specifications. For this work project, the EIPs consulted were:

- ERC-20. Token Standard. Describes the implementation of a standard API for tokens within smart contracts.Fabian Vogelsteller [2015]

[3] BitTorrent is a communication protocol for peer-to-peer file sharing, which enables users to distribute data and electronic files over the Internet in a decentralized manner. To send or receive files, users use a BitTorrent client on their Internet-connected computer

[4] Git is a version control system used to keep track of files and data by storing a tree of hashes and allowing distributed non linear collaboration trough the creation of repositories and branches.

[5] A Merkle DAG is a Merkle Tree in the form of a directed graph without directed cycles. For more information about DAG see DAG definition in 3.1.2.

- ERC-721. NFT Token Standard. The following standard allows for the implementation of a standard API for NFTs within smart contracts. This standard provides basic functionality to track and transfer NFTset al. [2018a].

- ERC-115. Multi Token Standard. A standard interface for contracts that manage multiple token types. A single deployed contract may include any combination of fungible tokens, non-fungible tokens or other configurations (e.g. semi-fungible tokens)et al. [2018b].

Since its creation, the NFT Standard, has been used for the Decentralized Autonomous Organization (DAO)s and systems have been created to provide tools to final users for the usage of unique assets. More of this work however has been implemented in the Open Blockchains.

2.4.1 NFT on Open Blockchains

Many initiatives have emerged as prototypes to explain to the general public the importance, relevance, and potential of NFT for storing and certifying digital unique assets. At the beginning pure art concepts or games on the Ethereum Blockchain have been very successful, and led to the generation of speculative markets with high volatility and fraud due to its economical and lucrative basis Steinwold [2019].

The most successful NFT projects speaking in terms of usability and profitability are:

- **Rare Pepes (2017)**. With Ethereum gaining prominence in early 2017, memes started to be traded there as well. A project named ”Peperium” was announced to be a “decentralized meme marketplace and Trading Card Game (TCG) that allowed anyone to create memes that live eternally on IPFS and Ethereum.”

- **Cryptopunks (2017)**. A first set of 10,000 unique computer-generated characters on the *Ethereum* Blockchain. The platform opted to let anyone with an Ethereum wallet claim a Cryptopunk for free.

  All 10,000 Cryptopunks were swiftly claimed and started a thriving secondary marketplace where people bought and sold them. By the time they were created no ERC-721 standard existed.

- **CryptoKitties (2017)**. It was the first NFT implementation to come mainstream. It is a blockchain-based virtual game that allows players to adopt, raise, and trade virtual cats with a unique genetic code embedded in the smart contract. No single asset would evolve or have same characteristics as the other and they would show new characteristics over time.

- **OpenSea (2017)**. Is an American online NFT marketplace aimed to work as a trading market for digital art and buyers. It is the most famous platform of its kind.

2.4.2 Permissioned Blockchains

Contrary to permissionless NFT platforms, permissioned Blockchains allow use of corporate data and business workflows to operate inter-system and inter-company wise.

Examples of NFT Permissioned Platforms are:

- **A Decentralized Framework for Patents and Intellectual Property as NFT in Blockchain Networks**Bamakan et al. [2021]. It proposes a system to implement store of intellectual property and patents in a corporate multi-shared NFT Blockchain platform. It proposes the architecture of a system with decentralized authentication and decentralize storage.

- **Design of extensible non-fungible token model in Hyperledger fabric**Hong et al. [2019] Presents an extensible NFT token model for supporting such assets in Hyperledger Fabric It also applies extensible NFTs such as document and signature tokens to a decentralized signature service.

Chapter 3

Approach

In this chapter a deeper exploration of the NFT-based systems is presented, same as the introduction and baseline of all systems. The analysis of the and main solution and proposed solutions will be presented.

3.1 Introduction

For the creation of this thesis work it was necessary to have a deep understanding of Blockchain Systems, distributed computation, DFSs, to study different consensus mechanisms and implementation of custom smart contracts able to extend the standards already available. All these technologies and the application of *Full Stack*[1] technologies create what is known today as the *Web 3.0*[2].

3.1.1 Blockchain

Blockchain-based distributed database systems are emerging technologies that aim to provide users with the necessary tools to perform tasks and computations without the assistance of an intermediate entity, to build self-trust systems where rules are agreed upon beforehand, and to create the basis for an ecosystem th can be

[1] It refers to developing both the front end (client-side) and the back end (server-side) components of a web application. Each component of the stack involves a special browser, user interface and server-side technology, frameworks and tools able to create reactive applications and fast asynchronous communications with the server. It includes database technologies, cloud computing, and DevOps frameworks.

[2] Web 3.0 will enable websites and apps to process information in a smart human-like manner through technologies such as machine learning (ML), big data, and decentralized ledger technology (DLT), etc

used by everyone. Moreover, users may interact democratically and be confident that committed transactions will be immutable, truthful, and irrefutable[3].

In its nature, a Blockchain is basically an immutable unbreakable data storage structure. It consists of a set of components that allow a set of data to be stored in small blocks and be linked by a Hash function to the Hash of its predecessor, forming a chain of blocks that all all the holding systems can verify and validate. This chain of blocks form a *Merkle Tree* structure; when the data inside any of the block changed by malicious users, the rest of the chain will not be approved by the community, thus resulting on a waste of resources and time. This happens because the hash of the *Merkle tree* will change and be different from the rest of the other nodes.

3.1.2 Blockchain Ledger

Figure 3.1 shows the main components of a Blockchain structure which form the Ledger. This basic components are generated and validated by the different nodes in the System and later allow other users to interact and submit transactions to it. Depending of the type of blockchain different consensus mechanisms are generated in order to insert and validate the transactions block by block. The components of the ledger are:

Block

Represents a set of stored transactions submitted by different users and approved by node validators. To be able to insert data into a single block, certain rules have to apply:

- A user connected to the system and willing to insert data (submit a transaction) needs to prove its identity with a public and private key, which will allow him to interact with its asset in order to update the ledger by sensing or receiving (update or submit). Once the user and its ownership are properly identified and approved respectively, the transaction enters in a ”data cache” where other transactions are waiting to be submitted in the ledger. There are several ways data can be validated and approved to be inserted into the ledger, but most commonly public Blockchains will request a transaction fee (paid in tokens) to commit the transaction. The fee allows the system to function as the machines in charge of maintaining and approving the transactions receive such value in exchange of the resources they use. Depending

[3] Bitcoin is the first and most famous example of a worldwide distributed blockchain system.

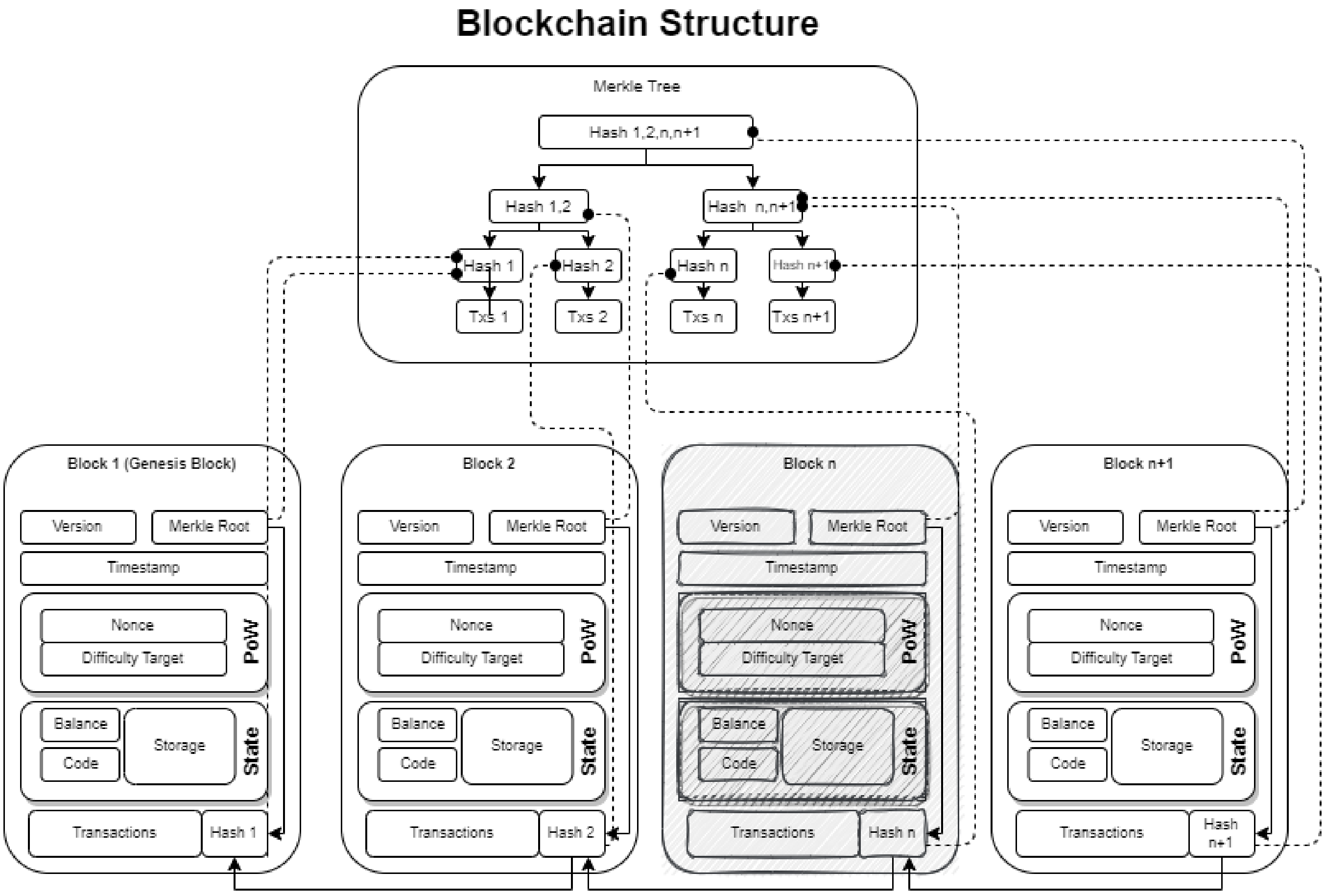


Figure 3.1: Structure of a PoW Blockchain ledger

on the amount paid in the fee, the transaction will take or not priority to be inserted in the next block. Once the block reaches a certain size or time, it is generated, hashes and send to other peers.

Version

Indicates the version that the particular block uses. There are three types of Blockchain versions.

- *Version 1.0 (cryptocurrency)*). It used a public ledger to store the data.
- *Version 2.0 (Smart Contracts)*). Indicates that embedded source code in the

block has been executed for the generation of the block.

- *Version 3.0 (Decentralized Application (DAPP)s).* It is used to create a decentralized structure, such as the Tor Browser.
- *Version 4.0 (Blockchain for Industry).* Used to create a scalable, affordable blockchain network so that more people can make use of it.

Timestamp

Used as proof that the particular block is used at what instance of a time. It is also used as a parameter to verify the authenticity of a block.

Consensus

A consensus is a fault-tolerant mechanism used to achieve agreement on a single data value or a single state of the network among distributed processes or multi-agent systems. It allows all parties to participate democratically and enable equal rights to validate and insert data in the Blockchain.

There are different consensus mechanismsBit [2018]:

- *PoW* One party proves to another which verifies that a particular computational effort has been expended. Subsequently, verification can be accomplished with minimal effort on the part of the verifying party. By requiring some work from a service requester, such as processing time by a computer, it was initially conceived to deter denial-of-service attacks and other forms of network abuse. In Bitcoin, it is used as a consensus mechanism for a permissionless decentralized network in which miners compete to append blocks and issue new currency with a success probability proportional to the computational effort expended. A key characteristic of proof-of-work schemes is their asymmetry[4]. Its design contains a built-in incentive structure that rewards validators by allocating computational capacity to the network with value in the form of money. PoW algorithms aim to deter data manipulation by establishing considerable energy and hardware-control requirements, thus expending large amounts of resources in the process.
- *Proof of Stake (PoS)* This method avoids the computational cost associated with proof-of-work schemes by selecting validators based on their holdings of

[4] The work (computation) that must be moderately complex, yet feasible by a computer (prover) but simple to verify by the service provider (verifier)

the associated cryptocurrency. The validators are rewarded for adding transactions to the block. PoS secures a system by requiring validators to possess some blockchain tokens to mount an attack. Because PoW does not require complex mathematical calculations, it is more energy-efficient.

- *Proof of Elapsed Time (PoET)* It is commonly used on a permissioned Blockchain. First, every node in the system must be identifiable and accepted into the network. Then, a standard certificate authority validates them. With PoET, the "timer" is different for each node. Each participant in the network is assigned a random amount of time to wait. The first participant to finish waiting gets to commit the following block to the blockchain.

- *Proof of Authority (PoA)* It is a spin on PoS consensus that addresses the risk of how participants in a network can value a stake. the consensus stakes the actual identities of the nodes in the system and aligns incentives by placing social capital at risk. A greater incentive is to act in the network's best interest if more of its net worth is lodged in a node, therefore wealthier participants could go Absence Without Official Leave (AWOL) as they can eat any financial loss they would receive for doing so. There also exist validation nodes, which stake their reputation on the network. As compensation, validators are the only nodes allowed to validate blocks. By identifying validators, PoA consensus becomes inherently centralized and best suited for private Blockchains and consortiums, such as a group of banks or insurance companies.

- *Byzantine Fault Tolerance (BFT)* Allows distributed system to reach consensus (agreement on the same value) even when some of the nodes in the network fail to respond or respond with incorrect information. The objective is to safeguard against the system failures by employing collective decision making when it is both correct and misleading. It aims to reduce influence of the faulty nodes. BFT is derived from Byzantine Generals' Problem[5].

- *Practical Byzantine Fault Tolerance (PBFT)* Designed to work efficiently in asynchronous systems. It is optimized for low overhead time. Its goal was to solve many problems associated with already available BFT solutions. It has a primary node and secondary nodes. These nodes work together to reach a

[5] The Byzantine Generals Problem is a term etched from the computer science description of a situation where involved parties must agree on a single strategy in order to avoid complete failure, but where some of the involved parties are corrupt and disseminating false information or are otherwise unreliable.

consensus, making this system one of the solutions to the Byzantine Generals Problem. The maximum number of faulty/malicious nodes cannot be equal to or greater than one-third of the total nodes in the system.

- *DAG* A directed graph is a DAG if and only if it can be topologically ordered, by arranging the vertices as a linear ordering that is consistent with all edge directions. To send a transaction, a node must validate two or more transactions that already took place. As more transactions are sent through the network, that system of checks and balances strengthens. The flow of data through this model allows the reduction of transactional fees, since they are approved as users contribute to the security of the network by confirming past transactions.

Number only used once (Nonce)

Is an arbitrary number that can be used just once in a cryptographic communication. Such number will never be repeated, and is the result that a machine has to come across in order to be the first on providing the hash of that number.

Difficulty Target

Is a measure of how difficult it is to mine a block in a blockchain for a particular cryptocurrency with PoW consensus. A high cryptocurrency difficulty means it takes additional computing power to verify transactions entered on a blockchain—a process called mining.

State

It is a pointer that systems of the blockchain system independently hold their own copy of the blockchain, and the current known ”state” is calculated by processing each transaction in order as it appears in the ledger. Transactions are bundled and delivered to each node in the form of a block. As new transactions are distributed throughout the network, they are independently verified and ”processed” by each node.

Transactions

Continuously set of data being created or updated in the ledger. Contains relevant information about the digital assets or smart contract instructions to operate and change the ledger.

Hash

It is the process of transforming any given key or a string of characters into a new value through the process of hashing. Shorter, fixed-length values can be represented by keys that enable finding and using the original data. A hash function generates new values according to a mathematical algorithm.. To prevent the conversion of hash back into the original key, a good hash always uses a one-way hashing algorithm.

Merkle Tree

Hash trees or Merkle trees are trees in which every leaf (node) is labelled with a cryptographic hash of a data block. The cryptographic hash of each inner node's label is included in its label. It allows efficient and secure verification of the contents of large data structures. Hash trees are a combination of hash lists and hash chains.

3.1.3 Additional generalities

All these components are basic to most of the Blockchain systems, regarding how nodes are assembled into the network depend mostly on the consensus mechanism adopted. for the purpose of this work the consensus aforementioned were deeply studied and incorporated in the proposal solution as it will be shown in the next sections.

3.1.4 Smart Contracts

There is one specific technology created by the Ethereum Foundation which allowed for the first time the embedding of source code into the Blockchain called *smart contract*.

A smart contract is a piece of structured code embedded into the Blockchain; it uses an Application Program Interface (API) to interact with the chain. Once in a block, it cannot be altered, but new versions and improvements can be reinserted. There is an implicit economical cost for submitting intelligent contracts into the chain and executing functions that will generate new blocks. Anyone willing to participate and be part of the contract must be a member of a DAO or Decentralized

Autonomous Corporation (DAC), use a public and key as a unique identity and a private key to sign the transaction being submitted. Generally speaking, smart contracts are public and can be easily visualized to know the conditions and rules that the involved parties will use to interact. Ethereum Network was the first blockchain to implement the generation and execution of Smart Contracts fully.

In the white paper Buterin [2014], it stipulates the basis for the usage of *Custom currencies* ans *colored coins* (first integrated in the Bitcoin Blockchain) and how they can be extended to the usage of custom tokens coded in a *Turing Complete*[6] programming language.

Smart Contracts have proved to be flexible and adaptable to deploy applications where trust and decentralization is vital. The technology is relatively new. Since 2015 many organizations have joined forces to include new programming language in different sets of ledgers. For the *Ethereum* Blockchain the standard language is known as *Solidity*Foundation [2022b]. With the arrival of other Systems, new programming languages where created whereas others like the *Hyperledger Foundation* implement *JavaScript*, *Java*, *Go* and *Solidity as well*.

They also allowed the creation of DAOs, where multiple parties could cooperate and collaborate in the benefit of the Blockchain ecosystem by agreeing on the rules that the smart contract was intended to execute depending on the conditions and situations the users will perform as a result of interacting with the DLT.

This phenomenon called the attention of the industry for its potential to cooperate and create immutable database systems, but adapted to an anonymous and permissioned approach.

Non-fungibility

NFT[7] in the blockchain brings tremendous possibilities to store value and a representation of physical or virtual objects. When the concept was exploited and implemented for the first time as a true standard in the Ethereum Environmentet al. [2018c], [8] the public realized of the potential the technology could have to generate

[6] A concept named after English mathematician and computer scientist Alan Turing- a system of data-manipulation rules (such as a computer's instruction set, a programming language, or a cellular automaton) is said to be ”Turing complete” or ”computationally universal” if it can be used to simulate any Turing machine.

[7] An object is fungible when and if it is identical to others and thus can be replaced without any loss (mutual interchangeability). On the other hand, an object is non-fungible if it posses unique properties, making it unequal to others and thus of a different transactional value than their reciprocals.

[8] Ethereum is another decentralized open source blockchain with smart contract functionality embedded since its conception. Conceived in 2013, its creator extended the potential that Bitcoin

value over the issuance of public certificates able to be verified by anyone to proof authenticity or ownership for predefined assets.

Although NFTs have skyrocketed from public Blockchains to demonstrate their potential with digital art (paintings, music, certification of authenticity, among others), little has been researched and implemented in the industry. But Non-fungibility has proven to be a key player for future technologies and mutual collaboration. Having a system with which companies can implicitly trust and collaborate offers tremendous possibilities for the development of knowledge and data protection.

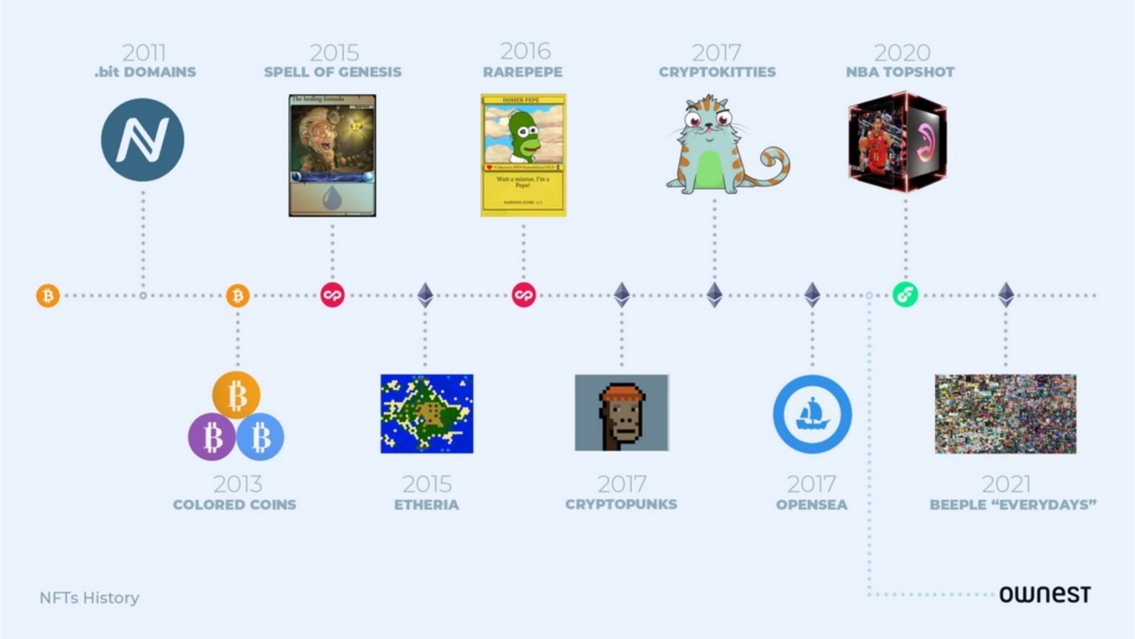


Figure 3.2: NFT Timeline, from the creation of Bitcoin domains, coloured coins to the new Ethereum and Altcoin derivatives Own [2021]

Industries can nowadays benefit from blockchain systems to record and track data in more trustful ways and allow inter-entity cooperation. However, the problem of privacy, ownership, security and finance emerges when trying to implement such systems on a public blockchain.

had as a decentralized system to enable the generation of digital contracts.

3.1.5 Hyperledger Fabric

Hyperledger Fabric is an open-source enterprise-grade permissioned DLT platform designed for use in enterprise contexts that delivers crucial differentiating capabilities over other popular distributed ledger or blockchain platforms. It was established under the Linux Foundation and currently holds the solid support of enterprises and developers for continuous improvement. Important features:

- Highly modular and configurable architecture, enabling innovation, versatility, and optimization for various industry use cases (banking, finance, insurance, healthcare, human resources, supply chain, digital music delivery, and others.).

- First distributed ledger platform to support chaincode[9] authored in a general-purpose programming language (Java, Go, and Node.js), facilitating their development without learning a new language or Domain-specific Language (DSL).

- Permissioned platform. Unlike public permissionless networks, the participants are known to each other. While the participants may not fully trust one another (they may be competitors in the same industry), a network can be operated under a governance model built off what trust exists between participants, such as a legal agreement or framework for handling disputes.

- Support for pluggable consensus protocols enables the platform to be more effectively customized to fit particular use cases and trust models. By default, Fabric implements BFT consensus fully but could be modified instead for another Crash Fault Tolerant (CFT) consensus protocol when fewer parties or organizations participate.

- Can leverage consensus protocols that do not require a native cryptocurrency to incent costly mining or fuel smart contract execution. Avoidance of a cryptocurrency reduces some significant risk/attack vectors, and the absence of cryptographic mining operations means that the platform can be deployed with roughly the exact operational cost as any other distributed system.

- Enables privacy and confidentiality of transactions and chaincode.

These differentiators make Fabric one of the better performing platforms available today regarding transaction processing and confirmation latency.

[9] In Hyperledger Fabric Smart Contracts are bettern known as ”chaincode”

Hyperledger Fabric Components

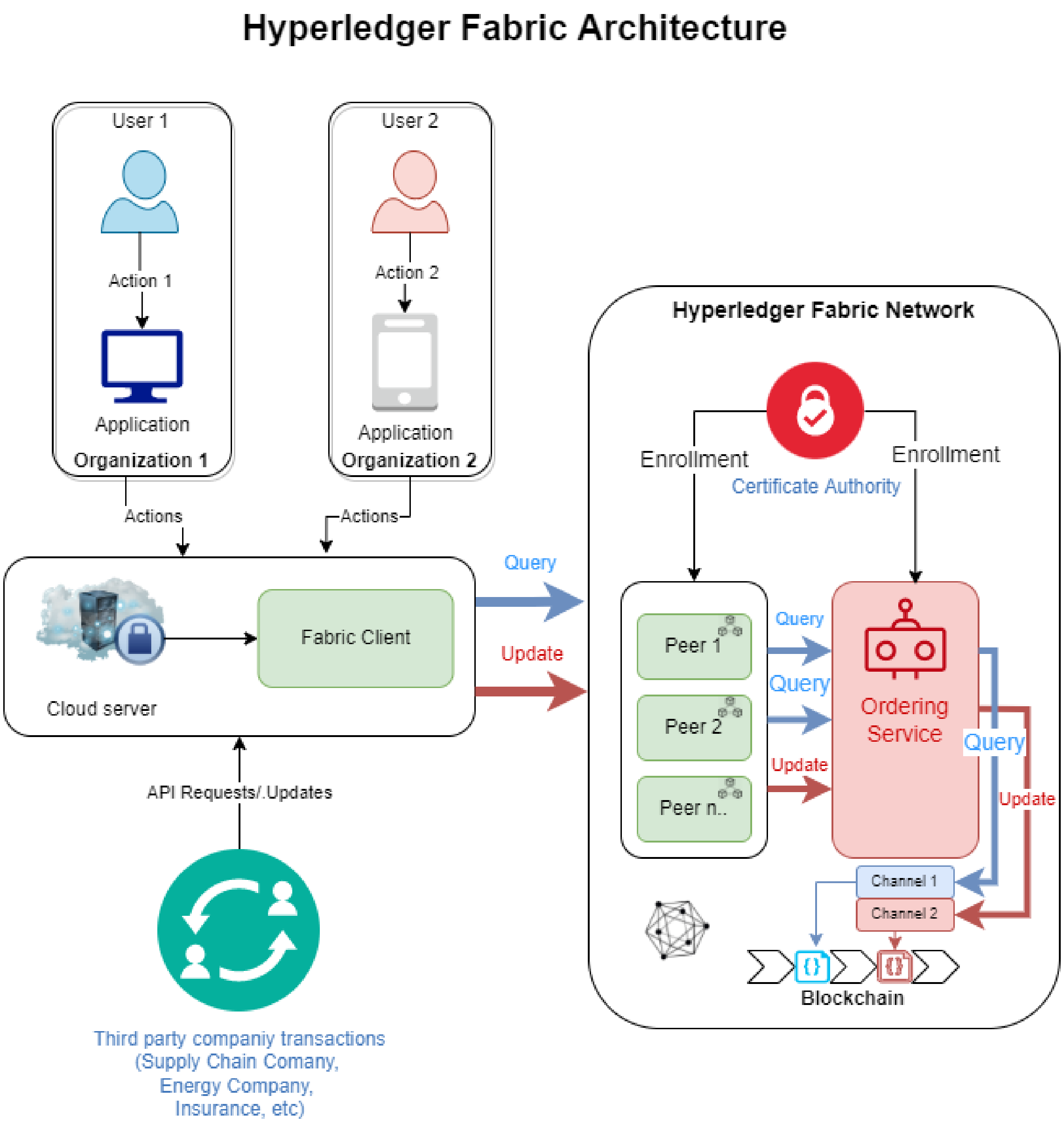


Figure 3.3: Hyperledger Fabric Architecture

- **Organization**. Virtual representation of a company/organization. Can define the areas and users that the organization will allow to interact with the

DLT

- **User**. Defines the entity that will directly interact with the ledger via *Fabric Client* through a Public Key Infrastructure (PKI) provided by a Certificate Authority (CA).

- **Application**. Is a custom software used by the organization and able to communicate with the ledger. The application will communicate with the ledger by sending actions to the *Fabric Client*.

- **Cloud server**. Infrastructure that hosts and provides the *Fabric Client* that can directly communicate with the ledger.

- **Fabric Client**. Application built by Hyperledger or custom code able to send query or update instructions to the ledger by invoking chaincode functions.

- **Third-party companies**. Other organizations with custom applications or network infrastructure can also communicate with *Fabric client* whenever the CA grants access to the ledger.

- **Certificate Authority**. Authorization service provider which uses PKI-based certificates to network member organizations and their users. Issues one root certificate to each member and one enrollment certificate to each authorized user.

- **Peer**. Servers that host a copy of the blockchain. Peers belong to the Organizations, and different organizations can have more than zero peers. The peers are coordinated by the ordering service and some of them will be selected as endorsers to validate chaincode transactions.

- **Ordering Service** is in charge of ordering the transactions. An orderer or set of orderer nodes form an ordering service. Because Fabric's design relies on deterministic consensus algorithms, any block validated by the peer is guaranteed to be final and correct. This architecture promotes finality by separating the endorsement of chaincode execution (which happens at the peers) from ordering, which provides advantages in performance and scalability as it eliminates bottlenecks.

- **Channel**. Is a private tunnel (subnet) of communication between one or more organizations through which the parties agree over one or more chaincode instructions. Whenever organizations join a party, no other external entity is able to see the interactions between then, ensuring external anonymity and inner transparency. Peers can join one or more channels. Each channel has its own chain of blocks.

- **Blockchain**. Is the ledger of the infrastructure, it exists in the peers and communicates via a single channel.

Hyperledger Fabric Workflow

The following steps form part of a typical Fabric workflow configuration:

1. A user invokes a chaincode execution through his application, which generates a transaction invocation. Client broadcasts the transaction invocation request to the Endorser peer

2. The Endorser peer checks the Certificate to validate the transaction. If verification is approved,it simulates the transaction by generating a response with a read-write set. Afterwards endorses the generated response using its own certificate. If the transaction fails a rejection response is sent.

3. The client receives the endorsed proposal responses from Endorsing Peers.

4. The client now sends the approved transaction to the orderer peer for this to be properly ordered and be included in a block.

5. The orderer node includes the transaction into a block.

6. The orderer node broadcasts the generated block to all Peers (to both Endorsing Peers and Committing Peers) on the relevant channel. Then, each Peer ensures that each transaction in the received block was signed by the appropriate Endorsing Peers. These individual peers then update their local ledger with the latest block. Thus all the network gets the ledger synced.

7. The Clients receive any subscribed events if any.

3.1.6 IPFS

IPFS is a distributed, Peer-to-Peer (P2P) file-sharing network enabling a high-scalability decentralized webBenet [2014]. Unlike common Internet sites that use

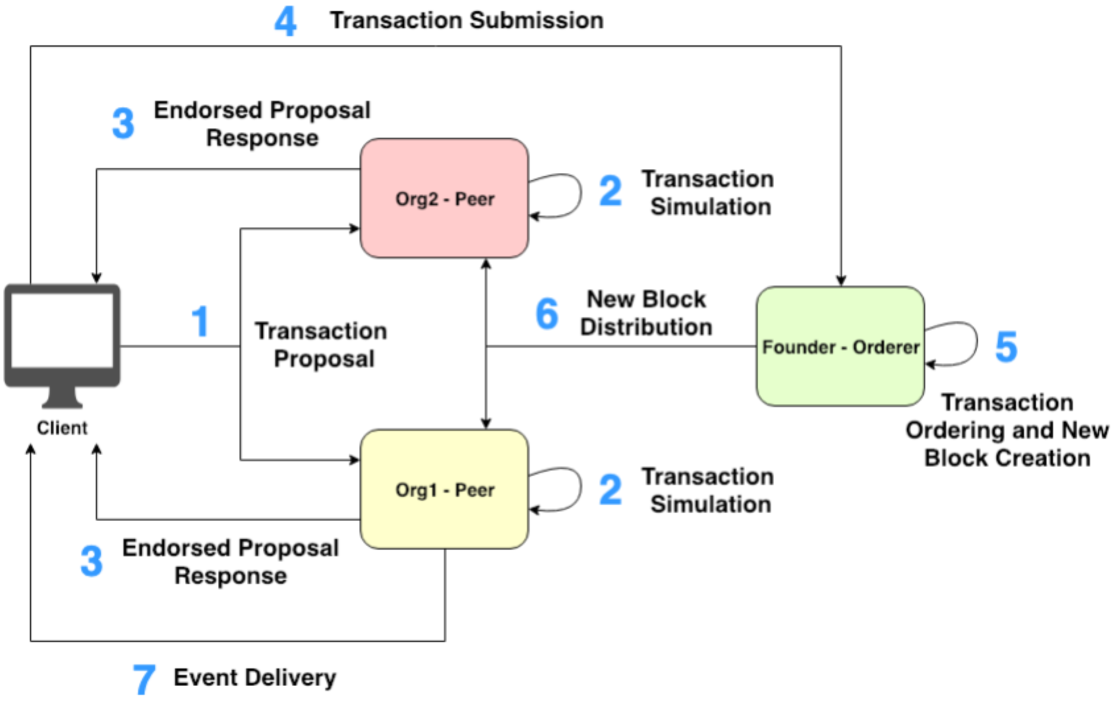


Figure 3.4: Hyperledger Fabric General Workflow

location-based address, IPFS stores information via content-based address and in the same manner as blockchain does, propagates the stored information in blocks to the connected peers. With this technology one can ensure that the stored data cannot be forged as the address is a Content Identifier (CID), which in essence is the root-hash of a Merkle Tree with additional metadata. Similar to blockchain systems, altering the information by even a single bit would drastically alter the hashing result.

Properties

- IPFS is made up of all the connected nodes, which can store data and make it accessible to anyone requesting it.
- If a user requests a file, a copy of the file is cached on their node. The more different users request that data, the more cached copies will exist. Subsequent requests for that file can be fulfilled by any node or combination of nodes owning the file, preserving that way the stored information.

- Because the data can be stored in pieces on many different computers, all those systems can feed parts of the data to its destination in parallel. This is intended to lower latency, reduce bandwidth, and avoid bottlenecks.

- As a result there is no focal point for hackers to attack.

- IPFS also offers the possibility to submit encrypted data.

- Data is stored in chunks of 256 KiloByte (KB), called IPFS objects. Files larger than that are split into as many IPFS objects as it takes to accommodate the file. One IPFS object per file contains links to all of the other IPFS objects that make up that file.

- Once a file is added to the IPFS network it is given a unique, 24-character hash CID, which is the identifier within the network. Recalculating the hash when the file is retrieved verifies the integrity of the file. If the check fails, the file has been modified.

- When files are legitimately updated, IPFS handles the versioning of files, meaning that new version of the file is stored along with the previous version. IPFS operates like a distributed file system, and this concept of versioning provides a degree of immutability to that file system.

- a Garbage collector will periodically remove cached IPFS objects unless they are pinned to be preserved.

The properties of this DFS provide meaningful advantages to work with Blockchain Systems since both work decentralized, trustless and in a set of hashed blocks.

3.2 Existing Approaches/Baselines

All the components of the system aforementioned form the baseline for this thesis work, however little has been done in permissioned Blockchain systems that allow data management as a NFT digital asset with IPFS. There is however more research on its usage for public Blockchain networks and other industry purposes as the mentioned below.

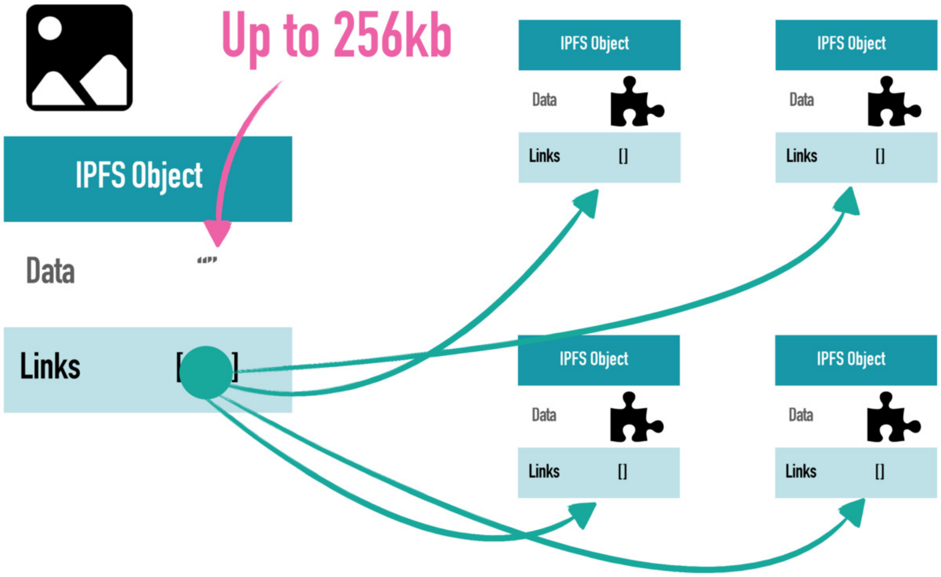


Figure 3.5: A file being stored in the IPFS network

e-Health

Healthcare systems maintain electronic medical records using the centralized storage models, potentially compromising user privacy. Potential threats include unauthorized access to critical information such as identity details, diseases from which a patient suffers, and misuse of patients' data. IPFS and blockchain technology, a distributed off-chain storage of medical data, can be created while preserving patient privacy.

- Kumar et al. [2020] proposes a framework to facilitate easy access to medical data by authorized entities while preserving consistency, integrity, and availability.

- Chen et al. [2021] suggests a system able to handle Electronic Health Records (EHR)s. for diabetes disease detection that provides an earlier detection of this disease by using various machine learning classification algorithms and securing the information. Blockchain, and IPFS are used to collect patient's health information via wearable sensor devices.

- Kumar et al. [2021] proposes in the same manner a decentralized system for medical data storage that allows the community to securely share information and remove it from central systems since the high cost of data breaches, cyber attacks and the implicit cost of restructuring the Information Technology (IT) infrastructure would prevent the progress in medical research and creation of new medical alternatives.

- VAHAKGupta et al. [2020] is an interesting project aiming to provide a low-latency, secure and reliable infrastructure via Ethereum smart contracts, 5-G and IPFS to allow communication and navigation of Unmanned Aerial Vehicle (UAV)s and improve the air-distribution of medical supplies in areas hard to reach or in crisis.

- HealChainNi et al. [2019] is a decentralized Data Management System (DMS) for Mobile Healthcare Using Consortium Blockchain, that aims to build a system where security prevails when patients share medical data wirelessly.

- Reen et al. [2019] is another system implementation to handle EHR criptographically and decentralized.

- Kayastha et al. [2021] Has implemented a case study in Nepal for an Ethereum and IPFS based Application model to record and share patient healdh information.

Patents and intellectual property

- Bamakan et al. [2022] intends to apply NFT-based patent framework in public Blockchain networks and divers DFS to the intellectual property to promote transparency and liquidity for innovators willing to commercialize their inventions or be funded.

- Agyekum et al. [2019] propose a digital media copyright and content protection using IPFS and Blockchain to expedite traditional digital copyright solidification and rights which are time consuming and labor-intensive using Hyperledger Fabric and the management of digital fingerprints for patents, ensuring immutability and provenance.

- Kalis and Belloum [2018] models a data Integrity and confidentiality framework using encryption, smart contracts and Apache Isis for the automatic audit trial.

Automotive

Nizamuddin and Abugabah [2021] proposes an IPFS/Ethereum blockchain-based system for the auto insurance sector to solve the problem of fraud and latency and bureaucracy that customers face whenever acquiring insurance for for vehicles. Insurance prevent the industry from progressing due to the liabilities and difficulties it presents in many situations to understand it.

Identity and Authorization

Battah et al. [2020] present a solution for Blockchain-based Multi-party authorization (MPA) for Accessing IPFS Encrypted Data using Ethereum smart contracts and proxy re-encryption algorithms. It incorporates reputation mechanisms in the smart contracts to rate the oracles based on their malicious and non-malicious behaviors. Thus, this system can solve insider's attack problem by ensuring that a single authority or party is not acting alone.

Tourism

Demirel et al. [2021] have created a model with Blockchain and IPFS integration for post pandemic economy for the tourism industry to fill the gap in the existing methods related to the use of the Internet of Things (IoT), devices with smart contracts without a need for intermediaries for the reservations and services secured by Blockchain. The authors have created a booking system with a unique smart contract between customers and hotels, including all services that a customer may need and elimination commission fees as well as reception costs.

Agriculture

Salah et al. [2019] have created a Blockchain-Based Soybean Traceability in Agricultural Supply Chain by utilizing smart contracts to govern and control all interactions and transactions among all the participants involved within the supply chain ecosystem. All transactions are recorded and stored in the ledger with links to IPFS and thus providing to all a high level of transparency and traceability into the supply chain ecosystem in a secure, trusted, reliable, and efficient manner.

3.3 Analysis

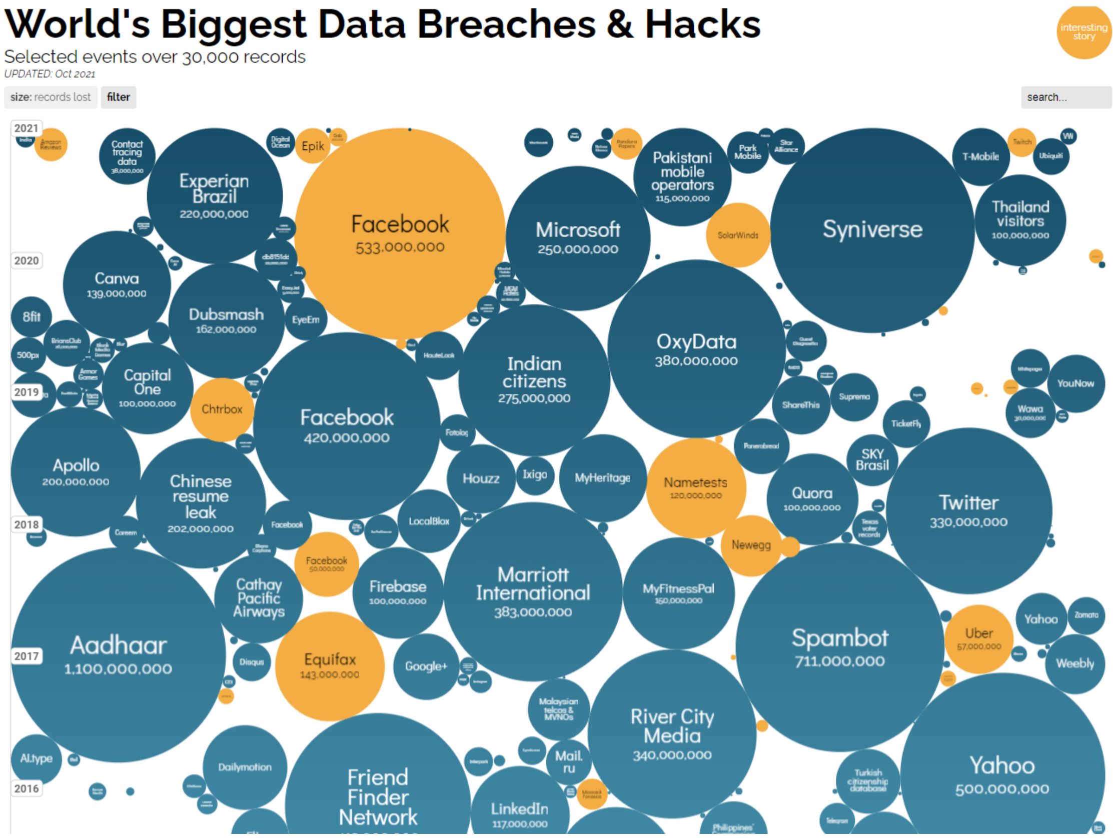
Ever since the world has been interconnected over the Internet, malicious parties have intended to take advantage over their computational resources and digital assets. As technologies improve and the society evolves into a digital world, attacks become more sophisticated, frequent and devastating. It is increasing month by month in an exponential ways, compromising governments and companies systems and data.

Such has been the risk and consequences of cyber attacks and data breaches that in 2021Wickr [2021]:

1. On average a data breach costs up to 8.64 million United States Dollar (USD).
2. Global Cybercrime costed over 6 trillion
3. Businesses fell victim of ransomware every 11 seconds.
4. Took up to 220 days to contain a data breach, with healthcare industry being the slowest to recover with over 320 days.
5. As more users perform remote works cybercriminals keep increasing their attacks over telecommuters and remote access pathways
6. Properly containing a data breach could have saved up to 1 million USD in less than 200 days
7. More than 8TeraByte (TB) of data were leaked.
8. A total of 270 major data breaches occurred, exposing 238 Million of records and 16 billion USD per day.

Moreover, in 2021 several Nordic companies were victim of important cyber attacks ,peaking in December 2021O'Dwye [2022]. Affected industries corresponded to the region's largest industrial, food and service providing sector. Affected companies were Vestas, Wind Systems, Amedia, Nortura and Nordic Choice Hotels

It is clear that with the current resources (both human and digital) is impossible to overcome these threats. In addition data has become a vital asset. Companies are now legally liable to protect it and ensure that is properly protected with state of the art technologies. Failing to do that could lead them to bankruptcy or serious fines which in some cases could be of irrecoverable damage, lost of reputation or even catastrophic disasters for the society.



s

Figure 3.6: Word’s Biggest data breaches and hacks occurred fro 2016 to October 2021 is beautiful [2021]

Hypothesis

With the implementation of a decentralized system in a Hyperledger Fabric and a chaincode implementing ERC-721 standard with IPFS network, it will be possible to manage and handle data between organizations in a safer and more secure way. Sharing information and ensuring that the non-repudation[10] principle remains consistent over the network no matter how many parties or users join the infrastructure.

[10] Assurance that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender’s identity, so neither can later deny having processed the information.

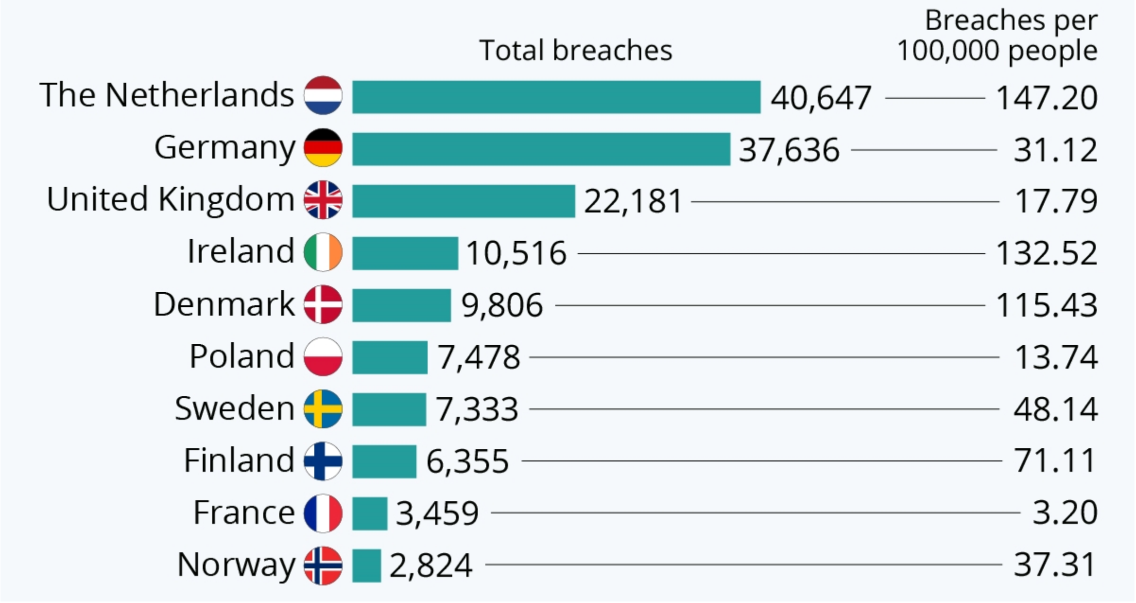


Figure 3.7: Top-10 GDPR-per-country data breaches notified per EEA jurisdiction from May 2018 to October 2020Statista [2020]

3.4 Proposed Solution

Having proposed the hypothesis to solve inter-organization data sharing and transmission through secure channels, the implementation of the system is shown in this section.

System technologies

The system gathers different technologies for the simulation

- **Docker**. Set of platform as a service products that use OS-level virtualization to deliver software in packages called containers.
- **Hyperledger Fabric version 2.2** is used in the system. It uses Docker technology to run and simulate the system. it uses **Alpine Linux** as the Operating System (OS) repository.
- **Typescript**. A programming language superset of JavaScript developed and maintained by Microsoft. This code was used to develop the chaincode, back-end and front-end systems.

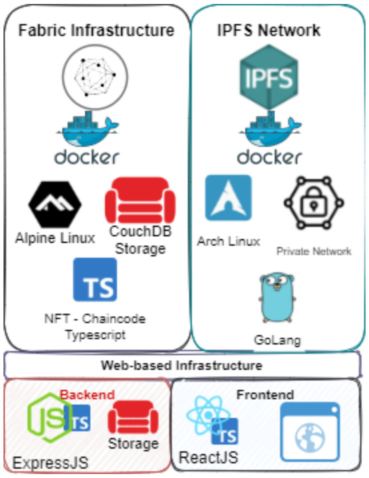


Figure 3.8: System technologies used

- **ExpressJS**. Back-end web application framework for Node.js designed for building web applications and APIs.
- **React**. Open-source front-end JavaScript library for building user interfaces based on UI components.
- **Apache CouchDB**. NoSQL document-oriented open-source database.
- **IPFS**. Decentralized file storage system used content-addressed capabilities configured as a private network. It has been built in **GO** language and **Alpine Linux** as OS. Uses **Docker** containers as a base image.

Proposed architecture

The components configured and assembled to integrate the infrastructure work in a Docker system.

- **Hyperledger Fabric**. A a set of shell scripts and docker files assemble the required infrastructure to build the DLT. The Fabric infrastructure has the following subcomponents:
    1. Organizations. Two organizations have been created. One organizations is build primarily to simulate the minting process of a NFT whereas the other can receive the minted token. The same process can also be made in the opposite way.

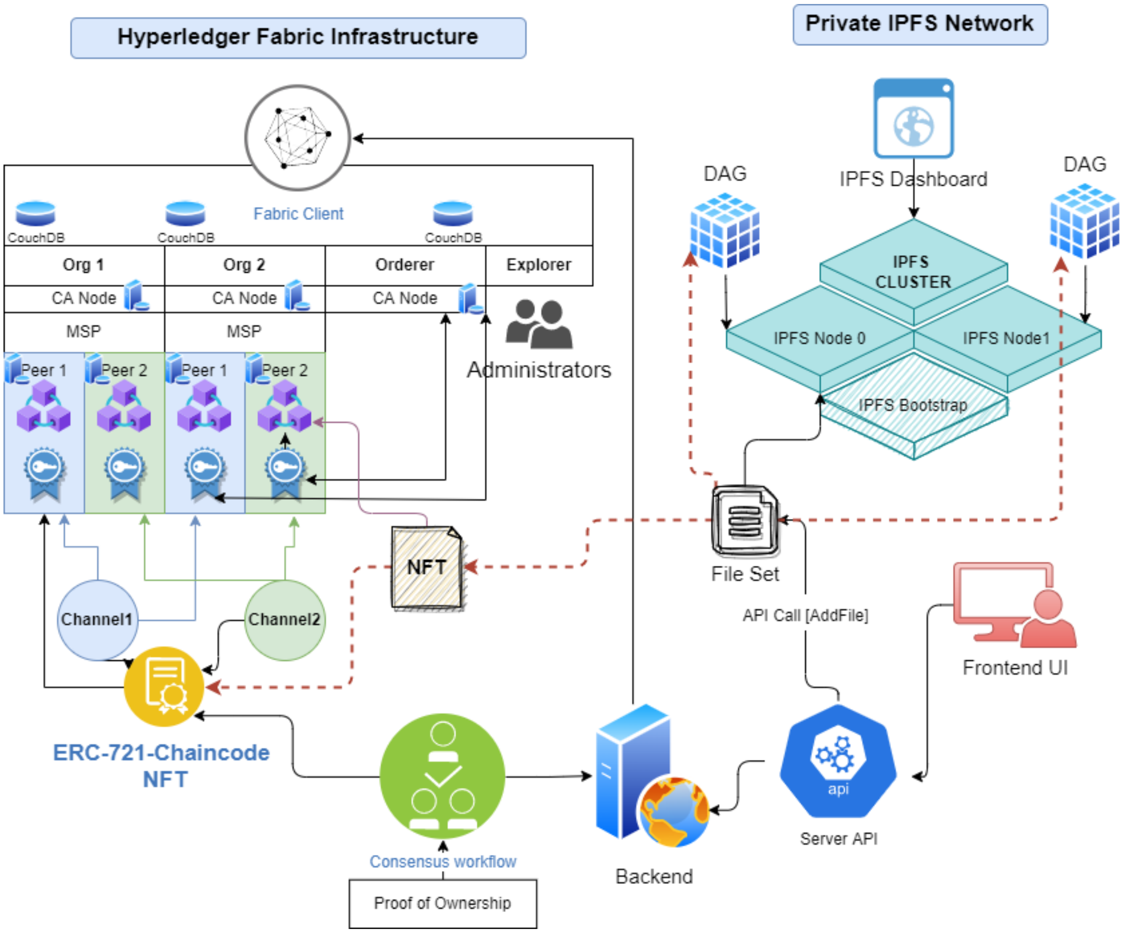


Figure 3.9: Proposed Architecture

2. Orderer node. Is the node in charge of sorting the transaction and creating the block. Once the block is created, it is forwarded to one of the organizations holding the Blockchain ledger. The orderer node does not hold a copy of the ledger but just coordinates and distributes the issued transactions.
3. Orderer CA. Is the node in charge of issuing and validating the certificates of the organizations and the wallet creation (based on the root certificates) for the users to interact with the system.
4. Channel. Organizations in Hyperledger can join and interact through a private channel. The channel is previously known by the two organizations and they participate by having a server that holds a version of the

ledger, a database server and a certificate authority server.

5. Chaincode. The smart contract used to manage the ledger transaction and mint digital assets as NFTs. The chaincode can be found in the appendix A.3.

- **Private IPFS network**. Consists of a set of Docker containers where each system holds a copy of the IPFS file system.

  – IPFS bootstrap node. In the same way as the Ordered node receives the blocks of a transaction and transmits it to the organization nodes, it receives the file or data and transmits it to all the interconnected nodes, ensuring data persistence among the network.

  – IPFS Dashboard. It works as the User Interface (UI) of the private network. Provides insights and data about the stored files and a preview of the data once a user can access through its content.

  – IPFS Nodes. Each organization can contribute to the system data repository by holding an IPFS server.

- **Back-end**. Server application that communicates with the Hyperledger network and extends its functionality by an API.

  – Connects organizations and corporate databases

  – Communicates with the CA servers to issue and hold certificates.

  – Communicates with IPFS private network API.

  – Holds a public Representational State Transfer (REST) API that provides all the functionality to register organizations, enroll users and mint NFTs.

  – Interacts with the Blockchain smart contract and retrieves the information contained in the ledger.

- **Front-end**. It serves as the UI for user interaction. Allows the simulation of the system where any user can create an organization, enroll users, mint a NFT and visualize them in the list.

Chapter 4

Experimental Evaluation

4.1 Use case

To simulate the system functionality a case for two persons from different organizations trying to issue and mint data, send it and making it available to one another.

4.2 Experimental Setup and Data Set

4.2.1 Scenario

Simulation of two organizations willing to share data with each other.

Organization 1

Represents an organization with important information gathered from external resources, expensive and difficult to obtain. Has the technology and infrastructure to generate data, but does not count with the know how neither infrastructure to process it or exploit it (similar to machine learning / big data scenarios).

User 'Minter' "Organization 1" Creates a user "Minter" as a company representative in charge of submitting the data and creating a NFT version of it inside the system. Then he can use the system to lend the data, enable a time frame visualization or transfer it.

Organization 2

Represents a technology company with skillful personal in data processing and domain area in the business of "Organization 1", but does not have the experience nor technology to gather it. Therefore it needs data from the aforementioned organization in order to create and expand their business.

User 'Receiver' "Organization 2" Creates a user "Receiver" as a company representative in charge of getting access to the NFT minted data by "Organization 1" via user "Minter" which will be used as the data source to perform machine learning training operations.

With the use of the NFT System framework these parties can cooperate and participate by trusting that the information is secure, has not been altered and can be easily verified by them and other parties.

4.3 Experimental Results

The complete simulation of the process is shown in this section.

4.3.1 System registering and user enrollment

The NFT system provides a friendly user interface to allow organizations to be registered into the Fabric network to use a specific channel. Prior to this process Fabric generated a CA with which they can identify as trusted entities and fair participants of the network.

The first time an organization registers itself to join a private channel, an admin account is created. The admin account then is able to configure organization parameters and new users with restricted access or different mining/system-usage capabilities. in the figure 4.1 can be visualized the process any organization will take to register itself as a valid participant with its corresponding representatives.

4.3.2 NFT Minting and data reading access

Once the organizations and users have been properly enrolled it the following steps must occur:

1. Minter access to the platform and selects the "Mint NFT option.
2. Minter selects a file and completes the form metadata.
3. Minter clocks on "submit" button to create the token.

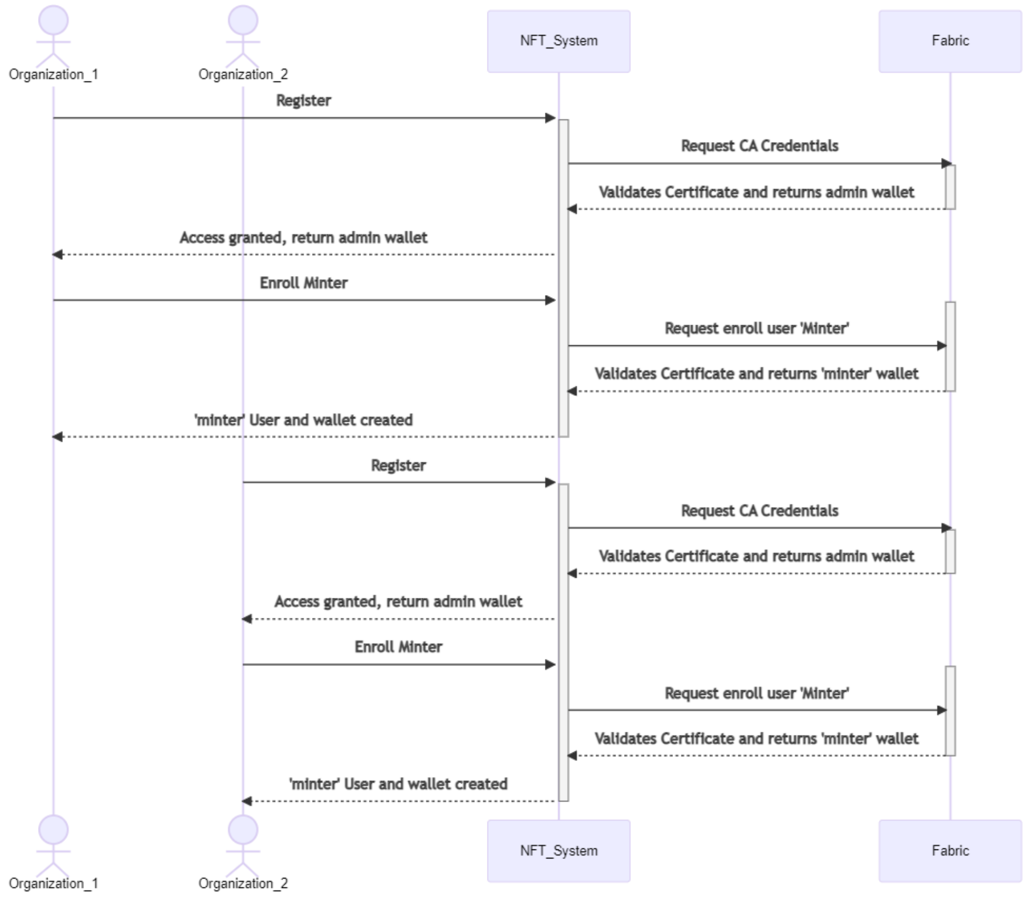

Organization_1
Organization_2
NFT_System
Fabric
Register
Request CA Credentials
Validates Certificate and returns admin wallet
Access granted, return admin wallet
Enroll Minter
Request enroll user 'Minter'
Validates Certificate and returns 'minter' wallet
'minter' User and wallet created
Register
Request CA Credentials
Validates Certificate and returns admin wallet
Access granted, return admin wallet
Enroll Minter
Request enroll user 'Minter'
Validates Certificate and returns 'minter' wallet
'minter' User and wallet created
Organization_1
Organization_2
NFT_System
Fabric

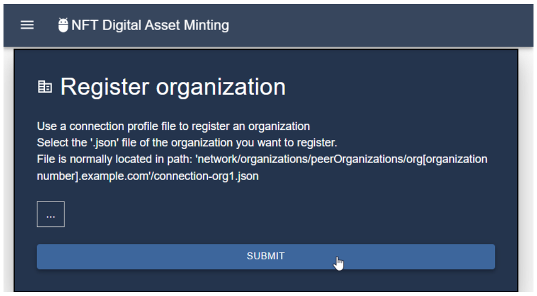
Figure 4.1: Sequence diagram of the Organization and user enrollment process in the system.

4. The System talks verifies that the file does not exist yet by:

    - Asking to add the data file to the IPFS network
    - Asking to Fabric if there is any token with the corresponding CID

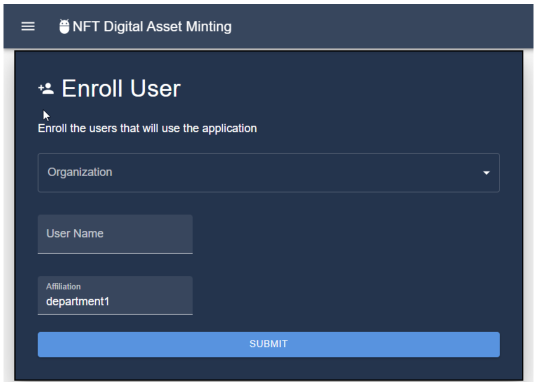
    For any of those cases to be true, the token will not be generated and an error will be returned.

5. Once everything is approved the token is stored in the DLT with the corre-



Use a connection profile file to register an organization
Select the '.json' file of the organization you want to register.
File is normally located in path: 'network/organizations/peerOrganizations/org[organization number].example.com'/connection-org1.json

Figure 4.2: UI Register organization.



Enroll the users that will use the application

Affiliation: department1

Figure 4.3: UI Enroll Users.

sponding file CID generated.

6. 'Minter' can now send the CID or additional token information to 'Receiver'. Then receiver can pull such information from the infrastructure. Because of its unicity purposes 'Receiver' will not be able to counterfeit or take ownership of the data unless explicitly stipulated and agreed by both parties.

4.3.3 NFT Transfer

It is possible through the REST API to transfer the generated tokens from one owner to the other. This process will acknowledge that the new user is the new owner of the NFT. It is possible then to track the chain of ownership.

4.3.4 NFT Burning

The system also supports burning an NFT, which basically blocks the token and flags it as unusable. if that happens, burned NFT and CID data cannot be accessed via the platform.

IT is possible, however to access the data via the IPFS server.

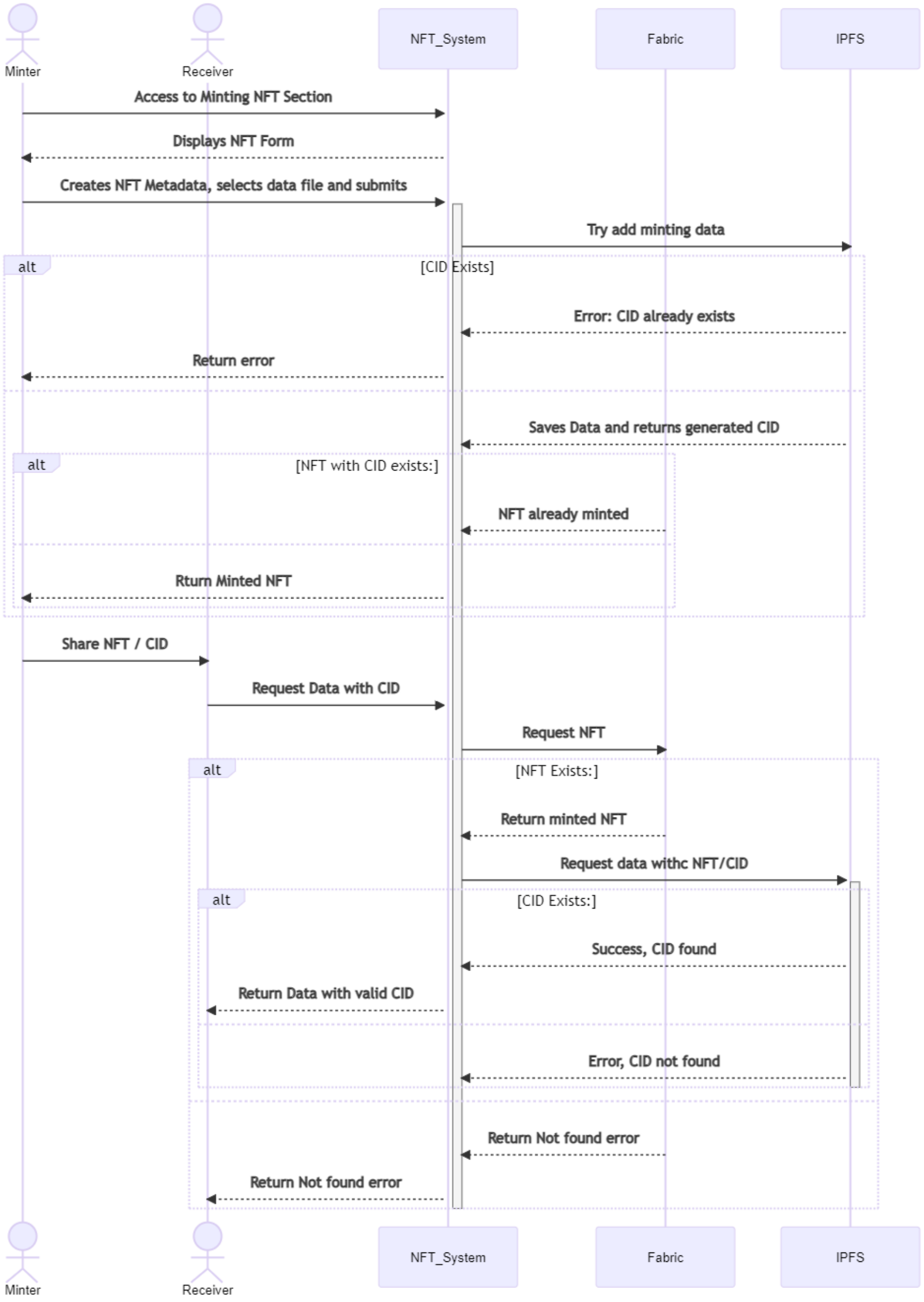
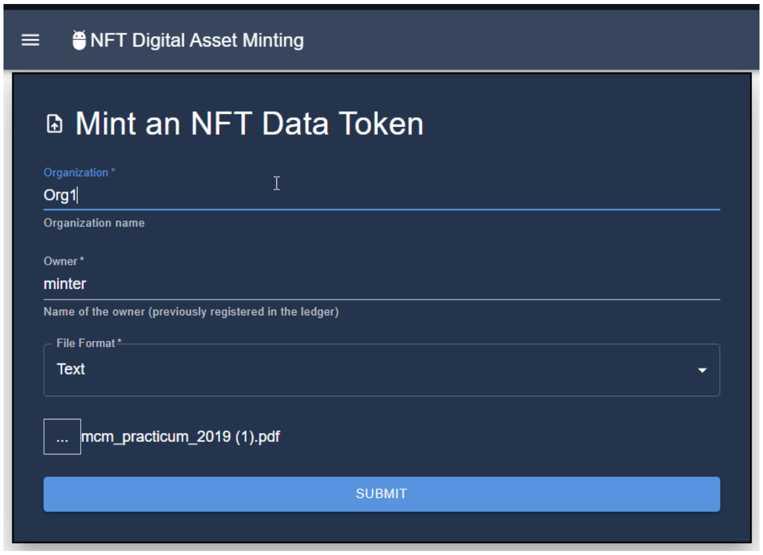


Figure 4.4: Sequence diagram to mint a NFT



Mint an NFT Data Token

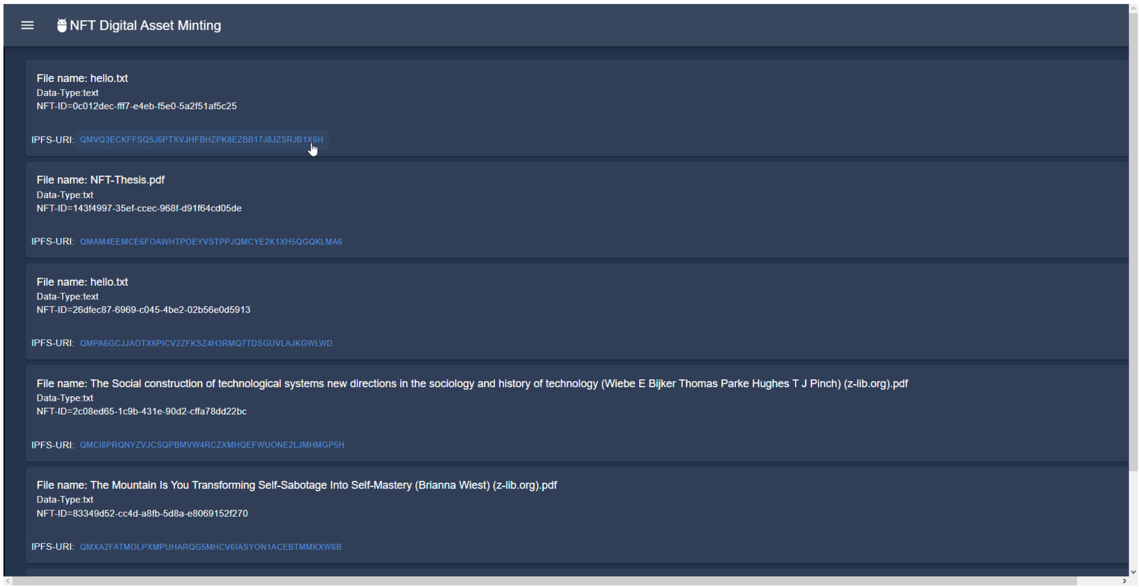
Organization *
Org1
Organization name

Owner *
minter
Name of the owner (previously registered in the ledger)

File Format *
Text

mcm_practicum_2019 (1).pdf

Figure 4.5: Front-end showing minting process.



File name: hello.txt
Data-Type:text
NFT-ID=0c012dec-fff7-e4eb-f5e0-5a2f51af5c25

IPFS-URI: QMVQ3ECKFFSQ5J6PTXVJHFBHZPK8EZBB17J8JZSRJB1X6H

File name: NFT-Thesis.pdf
Data-Type:txt
NFT-ID=143f4997-35ef-ccec-968f-d91f64cd05de

IPFS-URI: QMAM4EEMCE6FOAWHTPOEYVSTPPJQMCYE2K1XH5QGQKLMA6

File name: hello.txt
Data-Type:text
NFT-ID=26dfec87-6969-c045-4be2-02b56e0d5913

IPFS-URI: QMPA6GCJJAOTX6PICV2ZFKSZ4H3RMQ7TDSGUVLAJKGWLWD

File name: The Social construction of technological systems new directions in the sociology and history of technology (Wiebe E Bijker Thomas Parke Hughes T J Pinch) (z-lib.org).pdf
Data-Type:txt
NFT-ID=2c08ed65-1c9b-431e-90d2-cffa78dd22bc

IPFS-URI: QMCI8PRQNYZVJCSQPBMVW4RCZXMHQEFWUONE2LJMHMGP5H

File name: The Mountain Is You Transforming Self-Sabotage Into Self-Mastery (Brianna Wiest) (z-lib.org).pdf
Data-Type:txt
NFT-ID=83349d52-cc4d-a8fb-5d8a-e8069152f270

IPFS-URI: QMXA2FATMOLPXMPUHARQG5MHCV6IASYON1ACEBTMMKXW6B

Figure 4.6: Front-end showing minted NFTs in the system.

Chapter 5

Discussion

The implementation of digital asset management through issuance of NFTs represents a milestone in the generation of decentralized secure frameworks for industrial applications.

The implicit security of Blockchain with Hyperledger management and the privacy such technology offers will allow different organizations to participate and cooperate securely, but not anonymously.

All parties will be able to acknowledge data ownership. If desired, data could be encrypted as well and managed trough additional smart contracts. In addition to this, IPFS network is able to control, distribute and manage the added data as a DFS. The final simulation of the environment allows testers, to acknowledge the workflow of the framework and further expand its capabilities in a modular way.

5.1 Results

The results of the built application indicate that it is potentially feasible to create decentralized systems specialized in data management and control for industrial purposes.

Chapter 6

Conclusions

6.1 Final comments

In line with the original hypothesis stated in 3.3 it is possible to confirm that the construction of decentralized systems is possible and will create yet unforeseen possibilities to manage information and provide:

- System scalability
- Modularity
- Self Governance by smart contract agreement
- Privacy and security
- Mutual cooperation

Systems such as this can be governed or not by a central authority. Depending on the business needs and industrial purposes different scenarios could be developed and simulated.

Key findings through the research and development of this thesis project are:

6.1.1 Blockchain and DLT

. Data integrity and consistency safeguarding. Decentralized system nowadays allow very easily to verify data origin, integrity and non-repudation principle establishes new ways of working and contributing for the development and research of technology.

6.1.2 Security

The created infrastructure has several layers of security and implicit elements to protect parties and their data from improperly accessing information.

CAs

Different parties can choose to rely over a central or multiple CAs. As long as there is a common agreement previously established by smart contracts and common consensus, it will be possible to create highly resistant and resilient systems to malicious attacks. The modularity of Blockchain allows the cooperation of parties in benefit of the whole system, therefore rejecting undesirable or suspicious behaviour.

Private channels

Private channels in Hyperledger Fabric allow organizations to create a subsystem inside the framework, where sub smart contracts can be created to allow privacy between a single, two or more entities in different regions of the world. Therefore, while every organization can hold a copy of the ledger and data, only those with the right privileges can be allowed to interact with the hidden rules.

IPFS offers several layers of security, data can be encrypted and directed by smart contract instructions and even by the security layers set into the built system.

6.1.3 Governance

Organizations can self in their best interest to preserve business functionality and choose the most suitable way to manage data. That being said, they will be able to choose which CA is the best, what rules in the smart contract can be implemented and under which conditions entities can mint or transfer NFTs.

6.2 Future Work

It is intended for anyone willing to extend and expand the functionality of this thesis work to move within the following points:

6.2.1 System integration

This thesis project can complement the work made by Rehman [2022], which explains how shared data can be used to perform workflows for different purposes such as Big data analytics and Machine learning. One key advantage of this implementation relies on the fact that participants will have no direct access to shared data itself, but to a framework where it can be processed and manipulated to generate different models and insights. When connecting both systems, it will be possible to create and encapsulate information, self data encryption can grant that even different organizations own a copy of the blockchain and IPFS network, they cannot read the information unless they do it directly from the proposed application.

6.2.2 NFT and Smart Contract extension

An extension of ERC-721 smart contract implemented in A.3 can be easily extended to create new business rules such as:

NFT delegated ownership and transfer

Not only one user, but multiple entities could possess a digital asset, sharing a percentage of such element. Therefore new rules can be suggested to interact, protect and manipulate information.

Consensus

New consensus mechanisms can be generated to incentivize the usage of the network. This project contains for example rules to rank information and create a reputation level for the organizations, which can increase the value of the minted NFT. In other scenarios can be possible to create digital ownership by data origin, geographical location and mechanisms of burning so it cannot be used by other parties.

Economics through Tokenization

A very interesting approach to explore is the creation of economic tokens integrated with the NFT system. Such tokens and in the same form that Ethereum Cryptocurrency token (ETH) does with the Ethereum platform, every piece of data can be linked to another unity of tokens where once transferred its value in tokens can be transferred as well. The token-value of data can increase as its ranking of

”valuable information” increases, or organization reputation does. Depending on those economic mechanisms, different companies could be able to generate royalties and incentivize the usage of the system. Furthermore.

Multi-system integration

Furthermore, this project can be integrated with other public Blockchain platforms, and allow the issuance or reading of Ethereum smart contracts, enabling its execution in the internal network. The possibilities are endless and adaptable to specific business needs.

List of Figures

Appendix A

Code and Instructions

A.1 File repository

The repository with the code to download the system and perform the simulation is available at: https://github.com/asahicantu/NFT-Thesis.

Figure A.1: Qr Code which will redirect to the Github Project stated in A.1
.

A.2 Instructions to run the code

To run the project follow the following steps:

A.2.1 Prerequisites

A Linux operating system or bash scripting shell is required. On a windows machine the usage of Windows Subsystem for Linux (WSL) (any Linux distribution) can help to run the project Docker Desktop installed (if using Windows with WSL make sure the option 'Use WSL 2 Based engine' or similar is selected).

A.2.2 Run the application

1. Clone the repository

   ```
   git clone https://github.com/asahicantu/NFT-Thesis.git
   ```

2. Move to the repository's directory and then to the network directory

   ```
   cd NFT-Thesis/network
   ```

3. Enable execution mode for all .sh (shell scripting files)

   ```
   find . -name "*.sh" -exec chmod +x {} \;
   ```

4. Run the network infrastructure

   ```
   ./network start
   ```

5. Confirm no error occurred

6. Run server application in a different terminal

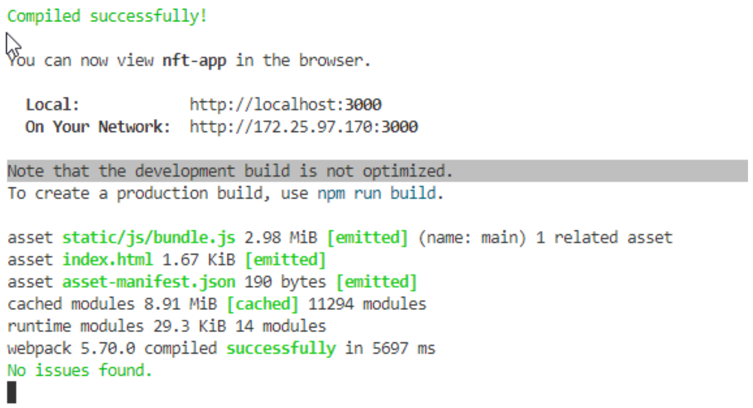
   ```
   cd ../web/server && npm install && npm run dev`
   ```

7. Confirm no error occurred

8. Run web application in a different terminal

   ```
   cd ../client && npm install && npm run start`
   ```

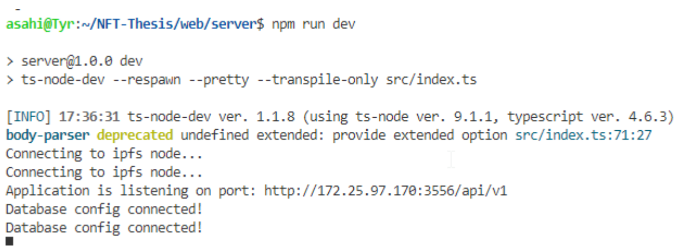
9. Confirm no error occurred



```
Compiled successfully!

You can now view nft-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.25.97.170:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

asset static/js/bundle.js 2.98 MiB [emitted] (name: main) 1 related asset
asset index.html 1.67 KiB [emitted]
asset asset-manifest.json 190 bytes [emitted]
cached modules 8.91 MiB [cached] 11294 modules
runtime modules 29.3 KiB 14 modules
webpack 5.70.0 compiled successfully in 5697 ms
No issues found.
```

Figure A.2: Network shell showing successful run

.

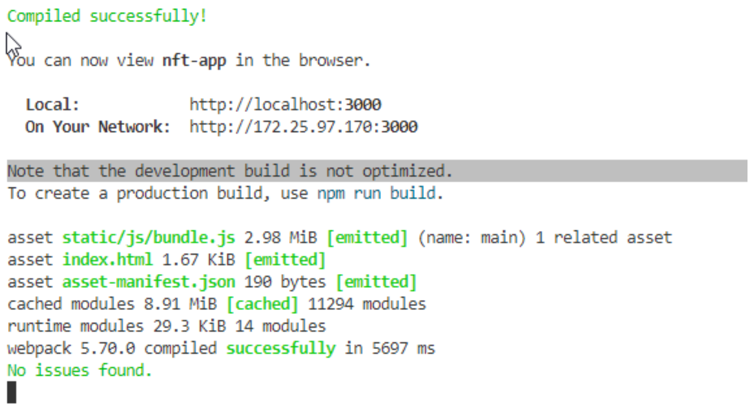


```
asahi@Tyr:~/NFT-Thesis/web/server$ npm run dev

> server@1.0.0 dev
> ts-node-dev --respawn --pretty --transpile-only src/index.ts

[INFO] 17:36:31 ts-node-dev ver. 1.1.8 (using ts-node ver. 9.1.1, typescript ver. 4.6.3)
body-parser deprecated undefined extended: provide extended option src/index.ts:71:27
Connecting to ipfs node...
Connecting to ipfs node...
Application is listening on port: http://172.25.97.170:3556/api/v1
Database config connected!
Database config connected!
```

Figure A.3: Server shell showing successful run

.

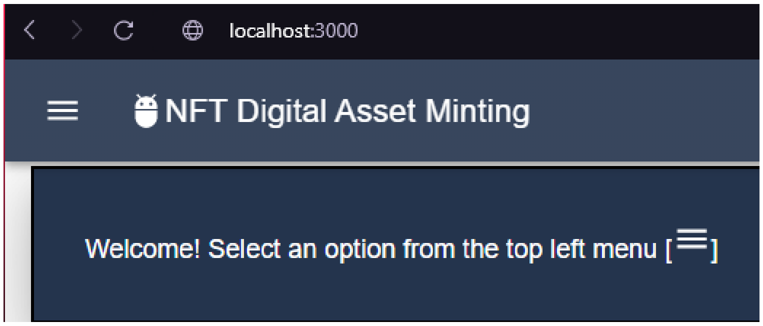
10. Open the application in a web browser by using: `http://localhost:3000`.
11. Confirm all steps were properly followed and no error occurred



```
Compiled successfully!

You can now view nft-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.25.97.170:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

asset static/js/bundle.js 2.98 MiB [emitted] (name: main) 1 related asset
asset index.html 1.67 KiB [emitted]
asset asset-manifest.json 190 bytes [emitted]
cached modules 8.91 MiB [cached] 11294 modules
runtime modules 29.3 KiB 14 modules
webpack 5.70.0 compiled successfully in 5697 ms
No issues found.
```

Figure A.4: Client shell showing successful run

.



Welcome! Select an option from the top left menu [☰]

Figure A.5: Main UI Page should be visible

.

A.3 NFT Chaincode

This is the chaincode or smart contract implementing ERC-721 Standard in Hyperledger Fabric. **File name:** tokenERC721Contract.ts

```
/*
SPDX-License-Identifier: Apache-2.0
*/
import { Context, Contract, Info, Transaction } from 'fabric-contract-api'
import { NFT } from 'common/nft'
import KV from './@types/KV'
```

```
@Info({ title: 'TokenERC721Contract', description: 'ERC721 SmartContract,
    implemented in TypeScript' })
export class TokenERC721Contract extends Contract {
    // Define objectType names for prefix
    balancePrefix: string = 'balance'
    nftPrefix: string = 'nft'
    uriPrefix: string = 'uri'
    approvalPrefix: string = 'approval'
    // Define key names for options
    nameKey: string = 'name'
    symbolKey: string = 'symbol'

    constructor() {
        super('TokenERC721Contract')
    }

    @Transaction(false)
    public async setLogLevel(ctx: Context, loglevel: string): Promise <
        void> {
        const logger = ctx.logger.setLevel(loglevel)
    }

    /**
    *
    * @param {Context} ctx the transaction context
    * @returns {Number} The number of non-fungible tokens present in the
        ledger
    */
    @Transaction(true)
    public async GetAllResults(ctx: Context, isHistory: boolean, owner:
        string): Promise<any> {
        const iterator = await
            ctx.stub.getStateByPartialCompositeKey(this.balancePrefix,
            [owner])
        let allResults = []
        let res = await iterator.next() as any
        while (!res.done) {
            if (res.value && res.value.value.toString()) {
                let jsonRes: any = {}
                console.log(res.value.value.toString('utf8'))
                if (isHistory && isHistory === true) {
```

```
                jsonRes.TxId = res.value.txId
                jsonRes.Timestamp = res.value.timestamp
                try {
                    jsonRes.Value =
                        JSON.parse(res.value.value.toString('utf8'))
                } catch (err) {
                    console.log(err)
                    jsonRes.Value = res.value.value.toString('utf8')
                }
            } else {
                jsonRes.Key = res.value.key
                try {
                    jsonRes.Record =
                        JSON.parse(res.value.value.toString('utf8'))
                } catch (err) {
                    console.log(err)
                    jsonRes.Record = res.value.value.toString('utf8')
                }
            }
            allResults.push(jsonRes)
        }
        res = await iterator.next()
    }
    iterator.close()
    return allResults
}

/**
 * BalanceOf counts all non-fungible tokens assigned to an owner
 *
 * @param {Context} ctx the transaction context
 * @param {String} owner An owner for whom to query the balance
 * @returns {Number} The number of non-fungible tokens owned by the
     owner, possibly zero
 */
@Transaction(false)
public async BalanceOf(ctx: Context, owner: string): Promise<number> {
    // There is a key record for every non-fungible token in the
        format of balancePrefix.owner.tokenId.
    // BalanceOf() queries for and counts all records matching
        balancePrefix.owner.*
    const iterator = await
        ctx.stub.getStateByPartialCompositeKey(this.balancePrefix,
        [owner])

    // Count the number of returned composite keys
    let balance = 0
    let result = await iterator.next()
    while (!result.done) {
        balance++
        result = await iterator.next()
    }
    return balance
}

/**
 * OwnerOf finds the owner of a non-fungible token
 *
 * @param {Context} ctx the transaction context
 * @param {String} tokenId The identifier for a non-fungible token
 * @returns {String} Return the owner of the non-fungible token
 */
@Transaction(false)
public async OwnerOf(ctx: Context, tokenId: string): Promise<string> {
    const nft = await this._readNFT(ctx, tokenId)
    const owner = nft.Owner
    if (!owner) {
        throw new Error('No owner is assigned to this token')
    }
    return owner
}

/**
 * TransferFrom transfers the ownership of a non-fungible token
 * from one owner to another owner
 *
 * @param {Context} ctx the transaction context
 * @param {String} from The current owner of the non-fungible token
 * @param {String} to The new owner
 * @param {String} tokenId the non-fungible token to transfer
 * @returns {Boolean} Return whether the transfer was successful or not
 */
```

```
@Transaction(true)
public async TransferFrom(ctx: Context, from: string, to: string,
    tokenId: string): Promise<boolean> {
    const sender = ctx.clientIdentity.getID()

    const nft = await this._readNFT(ctx, tokenId)

    // Check if the sender is the current owner, an authorized
        operator,
    // or the approved client for this non-fungible token.
    const owner = nft.Owner
    const approved = nft.ApprovedForTransfer == to
    const operatorApproval = await this.IsApprovedForAll(ctx, owner,
        sender)
    if (owner !== sender && !approved && !operatorApproval) {
        throw new Error('The sender is not allowed to transfer the
            non-fungible token')
    }

    // Check if `from` is the current owner
    if (owner !== from) {
        throw new Error('The from is not the current owner.')
    }

    // Clear the approved client for this non-fungible token
    nft.ApprovedForTransfer = undefined

    // Overwrite a non-fungible token to assign a new owner.
    nft.Owner = to
    const nftKey = ctx.stub.createCompositeKey(this.nftPrefix,
        [tokenId])
    await ctx.stub.putState(nftKey, Buffer.from(JSON.stringify(nft)))

    // Remove a composite key from the balance of the current owner
    const balanceKeyFrom =
        ctx.stub.createCompositeKey(this.balancePrefix, [from, tokenId])
    await ctx.stub.deleteState(balanceKeyFrom)

    // Save a composite key to count the balance of a new owner
    const balanceKeyTo =
        ctx.stub.createCompositeKey(this.balancePrefix, [to, tokenId])
    await ctx.stub.putState(balanceKeyTo, Buffer.from('\u0000'))

    // Emit the Transfer event
    const tokenIdInt = parseInt(tokenId)
    const transferEvent = { from: from, to: to, tokenId: tokenIdInt }
    ctx.stub.setEvent('Transfer',
        Buffer.from(JSON.stringify(transferEvent)))

    return true
}

/**
 * Approve changes or reaffirms the approved client for a non-fungible
     token
 *
 * @param {Context} ctx the transaction context
 * @param {String} approved The new approved client
 * @param {String} tokenId the non-fungible token to approve
 * @returns {Boolean} Return whether the approval was successful or not
 */
@Transaction(true)
public async Approve(ctx: Context, approved: string, tokenId: string):
    Promise<boolean> {
    const sender = ctx.clientIdentity.getID()

    const nft = await this._readNFT(ctx, tokenId)

    // Check if the sender is the current owner of the non-fungible
        token
    // or an authorized operator of the current owner
    const owner = nft.Owner
    const operatorApproval = await this.IsApprovedForAll(ctx, owner,
        sender)
    if (owner !== sender && !operatorApproval) {
        throw new Error('The sender is not the current owner nor an
            authorized operator')
    }

    // Update the approved client of the non-fungible token
    nft.ApprovedForTransfer = approved
    const nftKey = ctx.stub.createCompositeKey(this.nftPrefix,
        [tokenId])
    await ctx.stub.putState(nftKey, Buffer.from(JSON.stringify(nft)))

    // Emit the Approval event
    const tokenIdInt = parseInt(tokenId)
    const approvalEvent = { owner: owner, approved: approved, tokenId:
        tokenIdInt }
    ctx.stub.setEvent('Approval',
        Buffer.from(JSON.stringify(approvalEvent)))

    return true
}

/**
 * SetApprovalForAll enables or disables approval for a third party
     ("operator")
 * to manage all of message sender's assets
 *
 * @param {Context} ctx the transaction context
 * @param {String} operator A client to add to the set of authorized
     operators
 * @param {Boolean} approved True if the operator is approved, false
     to revoke approval
 * @returns {Boolean} Return whether the approval was successful or not
 */
@Transaction(true)
public async SetApprovalForAll(ctx: Context, operator: string,
    approved: boolean): Promise<boolean> {
    const sender = ctx.clientIdentity.getID()

    const approval = { owner: sender, operator: operator, approved:
        approved }
    const approvalKey =
        ctx.stub.createCompositeKey(this.approvalPrefix, [sender,
        operator])
    await ctx.stub.putState(approvalKey,
        Buffer.from(JSON.stringify(approval)))

    // Emit the ApprovalForAll event
    const approvalForAllEvent = { owner: sender, operator: operator,
        approved: approved }
    ctx.stub.setEvent('ApprovalForAll',
        Buffer.from(JSON.stringify(approvalForAllEvent)))

    return true
}

/**
 * GetApproved returns the approved client for a single non-fungible
     token
 *
 * @param {Context} ctx the transaction context
 * @param {String} tokenId the non-fungible token to find the approved
     client for
 * @returns {Object} Return the approved client for this non-fungible
     token, or null if there is none
 */
@Transaction(false)
public async GetApproved(ctx: Context, tokenId: string):
    Promise<string | object | undefined> {
    const nft = await this._readNFT(ctx, tokenId)
    return nft.ApprovedForTransfer
}

/**
 * IsApprovedForAll returns if a client is an authorized operator for
     another client
 *
 * @param {Context} ctx the transaction context
 * @param {String} owner The client that owns the non-fungible tokens
 * @param {String} operator The client that acts on behalf of the owner
 * @returns {Boolean} Return true if the operator is an approved
     operator for the owner, false otherwise
 */
@Transaction(false)
public async IsApprovedForAll(ctx: Context, owner: string, operator:
    string): Promise<boolean> {
    const approvalKey =
        ctx.stub.createCompositeKey(this.approvalPrefix, [owner,
        operator])
    const approvalBytes = await ctx.stub.getState(approvalKey)
    let approved
    if (approvalBytes && approvalBytes.length > 0) {
        const approval = JSON.parse(approvalBytes.toString())
        approved = approval.approved
    } else {
        approved = false
    }

    return approved
}

// ============== ERC721 metadata extension ===============

/**
 * Name returns a descriptive name for a collection of non-fungible
    tokens in this contract
 *
 * @param {Context} ctx the transaction context
 * @returns {String} Returns the name of the token
 */
@Transaction(false)
public async Name(ctx: Context): Promise<string> {
    const nameAsBytes = await ctx.stub.getState(this.nameKey)
    return nameAsBytes.toString()
}

/**
 * Symbol returns an abbreviated name for non-fungible tokens in this
    contract.
 *
 * @param {Context} ctx the transaction context
 * @returns {String} Returns the symbol of the token
*/
@Transaction(false)
public async Symbol(ctx: Context): Promise<string> {
    const symbolAsBytes = await ctx.stub.getState(this.symbolKey)
    return symbolAsBytes.toString()
}

/**
 * TokenURI returns a distinct Uniform Resource Identifier (URI) for a
    given token.
 *
 * @param {Context} ctx the transaction context
 * @param {string} tokenId The identifier for a non-fungible token
 * @returns {String} Returns the URI of the token
*/
@Transaction(false)
public async TokenURI(ctx: Context, tokenId: string): Promise<string> {
    const nft = await this._readNFT(ctx, tokenId)
    return nft.URI
}

@Transaction(false)
public async Token(ctx: Context, tokenId: string): Promise<NFT> {
    const nft = await this._readNFT(ctx, tokenId)
    return nft
}
/**
* Tokens returns all non-fungible tokens assigned to an owner
*
* @param {Context} ctx the transaction context
* @param {String} owner An owner for whom to query the balance
* @returns {Array<any>>} The number of non-fungible tokens owned by
    the owner, possibly zero
*/
@Transaction(false)
public async Tokens(ctx:Context, owner: string): Promise<Array<NFT>> {
    let tokens = new Array<NFT>()
    // There is a key record for every non-fungible token in the
        format of balancePrefix.owner.tokenId.
    // TokensOf() queries for all records matching
        balancePrefix.owner.* and returns all of them
    const iterator = await
        ctx.stub.getStateByPartialCompositeKey(this.balancePrefix,
        [owner])
    let result = await iterator.next()
    while (!result.done) {
        //const nft = result
        //nfts.push(result.value)
        //ctx.logger.getLogger().log('INFO', result.value)
        //console.log(result.value)
        var val = result.value as KV
        const nftKey = ctx.stub.createCompositeKey(this.balancePrefix,
            [owner])
        var id = val.key.replace(nftKey,'')
        id = id.replace(/\u0000/g, '')
        let nft = await this._readNFT(ctx, id)
        tokens.push(nft)
        result = await iterator.next()
    }
    return tokens
}

@Transaction(false)
public async TokenIds(ctx: Context, owner: string):
    Promise<Array<string>> {
    let tokenIds = new Array<string>()
    // There is a key record for every non-fungible token in the
        format of balancePrefix.owner.tokenId.
    // TokensOf() queries for all records matching
        balancePrefix.owner.* and returns all of them
    const iterator = await
        ctx.stub.getStateByPartialCompositeKey(this.balancePrefix,
        [owner])
    let result = await iterator.next()
    while (!result.done) {
        var val = result.value as KV
        tokenIds.push(val.key)
        result = await iterator.next()
    }
    return tokenIds
}

@Transaction(true)
public async Rate(ctx: Context, tokenId: string, organization:
    string,rank:string): Promise<NFT> {
    const nft = await this._readNFT(ctx, tokenId)
    if (!nft.RankerOrganizations.includes(organization)) {
        nft.RankerOrganizations.push(organization)
        let rankInt = parseInt(rank)
        nft.Weight = (rankInt / 5 * nft.Weight )
    }
    return nft
```

```
}

// ============== ERC721 enumeration extension ===============
/**
 * TotalSupply counts non-fungible tokens tracked by this contract.
 *
 * @param {Context} ctx the transaction context
 * @returns {Array<NFT>} Returns a count of valid non-fungible tokens
    tracked by this contract,
 * where each one of them has an assigned and queryable owner.
 */
@Transaction(false)
public async TotalSupply(ctx: Context): Promise<Array<NFT>> {
    // There is a key record for every non-fungible token in the
        format of nftPrefix.tokenId.
    // TotalSupply() queries for and counts all records matching
        nftPrefix.*
    const iterator = await
        ctx.stub.getStateByPartialCompositeKey(this.nftPrefix, [])
    let results = []
    // Count the number of returned composite keys
    let result = await iterator.next()
    while (!result.done) {
        if (result.value) {
            var nft = JSON.parse(result.value.value.toString()) as NFT
            results.push(nft)
        }
        result = await iterator.next()
    }
    return results
}
/**
 * TotalSupplyCount counts non-fungible tokens tracked by this
    contract.
 *
 * @param {Context} ctx the transaction context
 * @returns {Number} Returns a count of valid non-fungible tokens
    tracked by this contract,
 * where each one of them has an assigned and queryable owner.
 */
@Transaction(false)
```

```
public async TotalSupplyCount(ctx: Context): Promise<number> {
    // There is a key record for every non-fungible token in the
        format of nftPrefix.tokenId.
    // TotalSupply() queries for and counts all records matching
        nftPrefix.*
    const iterator = await
        ctx.stub.getStateByPartialCompositeKey(this.nftPrefix, [])

    // Count the number of returned composite keys
    let totalSupply = 0
    let result = await iterator.next()
    while (!result.done) {
        totalSupply++
        result = await iterator.next()
    }
    return totalSupply
}
// ============== Extended Functions for this sample ===============

/**
 * Set optional information for a token.
 *
 * @param {Context} ctx the transaction context
 * @param {String} name The name of the token
 * @param {String} symbol The symbol of the token
 */
@Transaction(true)
public async SetOption(ctx: Context, name: string, symbol: string):
    Promise<boolean> {

    // Check minter authorization - this sample assumes Org1 is the
        issuer with privilege to set the name and symbol
    const clientMSPID = ctx.clientIdentity.getMSPID()
    if (clientMSPID !== 'Org1MSP') {
        throw new Error('client is not authorized to set the name and
            symbol of the token')
    }
    await ctx.stub.putState(this.nameKey, Buffer.from(name))
    await ctx.stub.putState(this.symbolKey, Buffer.from(symbol))
   return true
}
```

```
/**
 * Mint a new non-fungible token
 *
 * @param {Context} ctx the transaction context
 * @param {String} tokenId Unique ID of the non-fungible token to be
    minted
 * @param {String} tokenURI URI containing metadata of the minted
    non-fungible token
 * @returns {Object} Return the non-fungible token object
*/

@Transaction(true)
public async Mint(ctx: Context, id: string, uri: string, format:
    string, owner: string, ownerOrg: string, filename:
    string,date:number): Promise<NFT> {

    // Check minter authorization - this sample assumes Org1 is the
        issuer with privilege to mint a new token
    const clientMSPID = ctx.clientIdentity.getMSPID()
    if (clientMSPID !== 'Org1MSP') {
        throw new Error('client is not authorized to mint new tokens')
    }

    // Check if the token to be minted does not exist
    let exists = await this._nftExistsById(ctx, id)
    if (exists) {
        throw new Error(`The token with Id ${id} is already minted.`)
    }
    exists = await this._nftExistsByUri(ctx, uri)
    if(exists){
        throw new Error(`The token with Uri ${uri} is already minted` )
    }
    const nftToken: NFT = {
        ID: id,
        URI: uri,
        FileFormat: format,
        Owner: owner,
        Organization: ownerOrg,
        FileName: filename,
        Weight: 0,
        ApprovedForTransfer: undefined,
        Date:date,
        RankerOrganizations: []
    }
    // Add a non-fungible token
    const nftKey = ctx.stub.createCompositeKey(this.nftPrefix,
        [nftToken.ID])
    await ctx.stub.putState(nftKey,
        Buffer.from(JSON.stringify(nftToken)))
    const uriKey = ctx.stub.createCompositeKey(this.uriPrefix,
        [nftToken.URI])
    await ctx.stub.putState(uriKey, Buffer.from(nftToken.ID))
    // A composite key would be balancePrefix.owner.tokenId, which
        enables partial
    // composite key query to find and count all records matching
        balance.owner.*
    // An empty value would represent a delete, so we simply insert
        the null character.
    const minter = ctx.clientIdentity.getID() // Get ID of submitting
        client identity
    const balanceKey = ctx.stub.createCompositeKey(this.balancePrefix,
        [minter, nftToken.ID])
    await ctx.stub.putState(balanceKey, Buffer.from('\u0000'))

    // Emit the Transfer event
    const transferEvent = { from: '0x0', to: minter, tokenId:
        nftToken.ID }
    ctx.stub.setEvent('Transfer',
        Buffer.from(JSON.stringify(transferEvent)))
    return nftToken
}

/**
 * Burn a non-fungible token
 *
 * @param {Context} ctx the transaction context
 * @param {String} tokenId Unique ID of a non-fungible token
 * @returns {Boolean} Return whether the burn was successful or not
 */
@Transaction(true)
public async Burn(ctx: Context, tokenId: string): Promise<boolean> {
    const owner = ctx.clientIdentity.getID()

    // Check if a caller is the owner of the non-fungible token
    const nft = await this._readNFT(ctx, tokenId)
    if (nft.Owner !== owner) {
        throw new Error(`Non-fungible token ${tokenId} is not owned by
            ${owner}`)
    }

    // Delete the token
    const nftKey = ctx.stub.createCompositeKey(this.nftPrefix,
        [tokenId])
    await ctx.stub.deleteState(nftKey)

    // Remove a composite key from the balance of the owner
    const balanceKey = ctx.stub.createCompositeKey(this.balancePrefix,
        [owner, tokenId])
    await ctx.stub.deleteState(balanceKey)

    // Emit the Transfer event
    const tokenIdInt = parseInt(tokenId)
    const transferEvent = { from: owner, to: '0x0', tokenId:
        tokenIdInt }
    ctx.stub.setEvent('Transfer',
        Buffer.from(JSON.stringify(transferEvent)))

    return true
}

/**
 * ClientAccountBalance returns the balance of the requesting client's
     account.
 * @param {Context} ctx the transaction context
 * @returns {Number} Returns the account balance
 */
@Transaction(false)
public async ClientAccountBalance(ctx: Context): Promise<number> {
    // Get ID of submitting client identity
    const clientAccountID = ctx.clientIdentity.getID()
    return this.BalanceOf(ctx, clientAccountID)
}
// ClientAccountID returns the id of the requesting client's account.
// In this implementation, the client account ID is the clientId
    itself.
// Users can use this function to get their own account id, which they
    can then give to others as the payment address
@Transaction(false)
public async ClientAccountID(ctx: Context): Promise<string> {
    // Get ID of submitting client identity
    const clientAccountID = ctx.clientIdentity.getID()
    return clientAccountID
}

private async _readNFT(ctx: Context, tokenId: string): Promise<NFT> {
    const nftKey = ctx.stub.createCompositeKey(this.nftPrefix,
        [tokenId])
    const nftBytes = await ctx.stub.getState(nftKey)
    if (!nftBytes || nftBytes.length === 0) {
        throw new Error(`The tokenId ${tokenId} is invalid. It does not
            exist`)
    }
    const nft = JSON.parse(nftBytes.toString())
    return nft as NFT
}

private async _nftExistsById(ctx: Context, tokenId: string):
    Promise<boolean> {
    const nftKeyId = ctx.stub.createCompositeKey(this.nftPrefix,
        [tokenId])
    const nftBytesById = await ctx.stub.getState(nftKeyId)
    return nftBytesById && nftBytesById.length > 0
}

private async _nftExistsByUri(ctx: Context, uri:string):
    Promise<boolean> {
    const nftKeyUri = ctx.stub.createCompositeKey(this.uriPrefix,
        [uri])
    const nftBytesByUri = await ctx.stub.getState(nftKeyUri)
    return nftBytesByUri && nftBytesByUri.length > 0
}
}
```

Bibliography

Satoshi Nakamoto. Bitcoin whitepaper. *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*, 2008.

Hyperledger Foundation. Hyperledger – open source blockchain technologies. https://www.hyperledger.org/, 2022a. (Accessed on 05/14/2022).

Hyperlerdger. A blockchain platform for the enterprise — hyperledger-fabricdocs main documentation. https://hyperledger-fabric.readthedocs.io/en/release-2.2/, 2022. (Accessed on 05/14/2022).

Juan Benet. Ipfs-content addressed, versioned, p2p file system. https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf, 2014.

Vitalik Buterin Fabian Vogelsteller. Eip-20: Token standard. https://eips.ethereum.org/EIPS/eip-20, 11 2015. (Accessed on 05/14/2022).

William Entriken et al. Eip-721: Non-fungible token standard. https://eips.ethereum.org/EIPS/eip-721, 01 2018a. (Accessed on 05/14/2022).

Witek Radomski Andrew Cooke et al. Eip-1155: Multi token standard. https://eips.ethereum.org/EIPS/eip-1155, 06 2018b. (Accessed on 05/14/2022).

Andrew Steinwold. The history of non-fungible tokens (nfts) | by andrew steinwold | medium. https://medium.com/@Andrew.Steinwold/the-history-of-non-fungible-tokens-nfts-f362ca57ae10, 10 2019. (Accessed on 05/15/2022).

Seyed Mojtaba Hosseini Bamakan, Nasim Nezhadsistani, Omid Bodaghi, and Qiang Qu. A decentralized framework for patents and intellectual property as nft in blockchain networks. 2021.

Sangwon Hong, Yoongdoo Noh, and Chanik Park. Design of extensible non-fungible token model in hyperledger fabric. In *Proceedings of the 3rd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, pages 1–2, 2019.

Daily Bit. 9 types of consensus mechanisms that you didn't know about. https://medium.com/the-daily-bit/9-types-of-consensus-mechanisms-that-you-didnt-know-about-49ec365179da, 04 2018. (Accessed on 05/15/2022).

Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf, 2014. (Accessed on 05/11/2022).

Ethereum Foundation. Introduction to smart contracts — solidity 0.8.13 documentation. https://docs.soliditylang.org/en/v0.8.13/introduction-to-smart-contracts.html, 2022b. (Accessed on 05/14/2022).

William Entriken et al. Eip-721: Non-fungible token standard. https://eips.ethereum.org/EIPS/eip-721, 01 2018c. (Accessed on 05/24/2022).

Ownest | nft month - history of nfts. https://ownest.io/en/news/history-of-nfts, 2021. (Accessed on 05/10/2022).

Randhir Kumar, Ningrinla Marchang, and Rakesh Tripathi. Distributed off-chain storage of patient diagnostic reports in healthcare system using ipfs and blockchain. In *2020 International Conference on COMmunication Systems NETworkS (COMSNETS)*, pages 1–5, 2020. doi: 10.1109/COMSNETS48256.2020.9027313.

Mengji Chen, Taj Malook, Ateeq Ur Rehman, Yar Muhammad, Mohammad Dahman Alshehri, Aamir Akbar, Muhammad Bilal, and Muazzam A. Khan. Blockchain-enabled healthcare system for detection of diabetes. *Journal of Information Security and Applications*, 58:102771, 2021. ISSN 2214-2126. doi: https://doi.org/10.1016/j.jisa.2021.102771. URL https://www.sciencedirect.com/science/article/pii/S221421262100020X.

Shivansh Kumar, Aman Kumar Bharti, and Ruhul Amin. Decentralized secure storage of medical records using blockchain and ipfs: A comparative analysis with future directions. *Security and Privacy*, 4(5):e162, 2021.

Rajesh Gupta, Arpit Shukla, Parimal Mehta, Pronaya Bhattacharya, Sudeep Tanwar, Sudhanshu Tyagi, and Neeraj Kumar. Vahak: A blockchain-based outdoor delivery scheme using uav for healthcare 4.0 services. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 255–260, 2020. doi: 10.1109/INFOCOMWKSHPS50562.2020.9162738.

Weiquan Ni, Xumin Huang, Junxing Zhang, and Rong Yu. Healchain: A decentralized data management system for mobile healthcare using consortium blockchain. In *2019 Chinese Control Conference (CCC)*, pages 6333–6338, 2019. doi: 10.23919/ChiCC.2019.8865388.

Gaganjeet Singh Reen, Manasi Mohandas, and S. Venkatesan. Decentralized patient centric e- health record management system using blockchain and ipfs. In *2019 IEEE Conference on Information and Communication Technology*, pages 1–7, 2019. doi: 10.1109/CICT48419.2019.9066212.

Mahesh Kayastha, Shakir Karim, Raj Sandu, and Ergun Gide. Ethereum blockchain and inter-planetary file system (ipfs) based application model to record and share patient health information: An exemplary case study for e-health education in nepal. In *2021 19th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–7, 2021. doi: 10.1109/ITHET50392.2021.9759580.

Seyed Mojtaba Hosseini Bamakan, Nasim Nezhadsistani, Omid Bodaghi, and Qiang Qu. Patents and intellectual property assets as non-fungible tokens; key technologies and challenges. *Scientific Reports*, 12(1):1–13, 2022.

Kwame Opuni-Boachie Obour Agyekum, Qi Xia, Yansong Liu, Hong Pu, Christian Nii Aflah Cobblah, Goodlet Akwasi Kusi, Hanlin Yang, and Jianbin Gao. Digital media copyright and content protection using ipfs and blockchain. In *International Conference on Image and Graphics*, pages 266–277. Springer, 2019.

Rosco Kalis and Adam Belloum. Validating data integrity with blockchain. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 272–277, 2018. doi: 10.1109/CloudCom2018.2018.00060.

Nishara Nizamuddin and Ahed Abugabah. Blockchain for automotive: An insight towards the ipfs blockchain-based auto insurance sector. *International Journal of Electrical & Computer Engineering (2088-8708)*, 11(3), 2021.

Ammar Ayman Battah, Mohammad Moussa Madine, Hamad Alzaabi, Ibrar Yaqoob, Khaled Salah, and Raja Jayaraman. Blockchain-based multi-party authorization for accessing ipfs encrypted data. *IEEE Access*, 8:196813–196825, 2020. doi: 10.1109/ACCESS.2020.3034260.

Engin Demirel, Seda Karagöz Zeren, and Kemal Hakan. Smart contracts in tourism industry: a model with blockchain integration for post pandemic economy. *Current Issues in Tourism*, pages 1–15, 2021.

Khaled Salah, Nishara Nizamuddin, Raja Jayaraman, and Mohammad Omar. Blockchain-based soybean traceability in agricultural supply chain. *IEEE Access*, 7:73295–73305, 2019. doi: 10.1109/ACCESS.2019.2918000.

Wickr. 10 data breach statistics for 2021 - wickr. https://wickr.com/10-data-breach-statistics-for-2021/, 2021. (Accessed on 05/16/2022).

Information is beautiful. World's biggest data breaches & hacks — information is beautiful. https://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/, 10 2021. (Accessed on 05/17/2022).

Gerard O'Dwye. Nordic companies targeted in wave of cyber attacks. https://www.computerweekly.com/news/252511965/Nordic-companies-targeted-in-wave-of-cyber-attacks, 01 2022. (Accessed on 05/16/2022).

Statista. The countries with the most gdpr data breaches | statista. https://www.statista.com/chart/20566/personal-data-breaches-notified-per-eea-jurisdiction/, 01 2020. (Accessed on 05/16/2022).

Ali Akbar Rehman. System for workflow design and execution on data shared between untrusting organizations for analytics. Master's thesis, University of Stavanger, Norway, 6 2022.

University
of Stavanger

4036 Stavanger
Tel: +47 51 83 10 00
E-mail: post@uis.no
www.uis.no