A Decentralized, Secure and Censorship Resistant Governance Protocol on Ethereum

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Oriol Rodríguez Setó

In partial fulfilment

of the requirements for the degree in

Telecommunications Technologies and Services Engineering

Advisor: Jose Luis Muñoz Tapia

Barcelona, June 2022

Abstract

A peer-to-peer voting system would allow a group of people to make collective decisions based on democratic principles without needing a regulator entity or third party. A system of this kind could allow democratic voting processes to be more secure, auditable, and trustworthy than the traditional processes currently being used in most democratic developed countries.

The requirements for a peer-to-peer voting system can be grouped into two main categories. First, a registration and access control system and second, a voting system. Both systems need to be peer-to-peer, secure against attackers and easily accessible.

This project proposes a solution intended to achieve a peer-to-peer system nature while keeping it affordable and scalable. The development has been done trying to achieve a voting process as frictionless as possible so that it can compete with traditional voting processes.

The solution has been developed by combining a set of peer-to-peer infrastructure technologies. This has allowed the implementation of all necessary functionalities involved in a collective decision-making process with the peer-to-peer system. At the same time, the system is simple and easy to use with an intuitive user interface and removing people's knowledge barriers to facilitate the use of these revolutionary technologies.

Resum

Un sistema de votació *peer-to-peer* permetria a un grup de persones prendre decisions col·lectives basades en principis democràtics sense la necessitat d'una entitat reguladora o tercera persona. Un sistema amb aquestes qualitats permet que el procés de votar democràticament sigui mes segur, auditable i fiable que els processos tradicionals que s'estan utilitzant a la majoria de països democràtics desenvolupats.

Els requeriments per el sistema de votació *peer-to-peer* poden agrupar-se en dos categories principals. Un sistema de registre i control d'accés i un sistema de votació. Els dos sistemes han d'ésser *peer-to-peer*, segurs contra atacs potencials i fàcilment accessibles.

Aquest projecte proposa una solució amb la intenció d'aconseguir un sistema *peer-to-peer* que alhora es manté assequible i escalable. El desenvolupament s'ha fet intentant aconseguir un procés de votacions amb la mínima fricció possible amb la intenció de poder competir amb processos tradicionals de recol·lecta de vots.

La solució s'ha desenvolupat combinant un conjunt d'infraestructures tecnològiques *peer-to-peer* que ha permet implementar totes les funcions necessàries involucrades en la presa de decisions col·lectiva. Al mateix temps, el sistema és simple i fàcil d'utilitzar amb una interfície d'usuari intuïtiva que elimina les barreres de coneixement dels usuaris per facilitar el ús d'aquestes tecnologies revolucionàries.

Resumen

Un sistema de votaciones *peer-to-peer* permitiría a un grupo de personas tomar decisiones colectivas basadas en principios democráticos sin la necesidad de una entidad reguladora o tercera persona. Un sistema con estas cualidades permite que el proceso de votar democráticamente sea más seguro, auditable y de fiar que los procesos de votación tradicionales que se utilizan actualmente en la mayoría de los países democráticos desarrollados.

Los requisitos para el sistema de votaciones entre iguales pueden agruparse en dos categorías principales. Primero, un sistema de registro y control de acceso, y segundo, un sistema de votación. Los dos sistemas tienen que ser sistemas *peer-to-peer*, ser seguros contra ataques y fáciles de acceder.

Este proyecto propone una solución con la intención de conseguir un sistema *peer-to-peer* que a la vez se mantiene asequible y escalable. El desarrollo se ha hecho intentando conseguir un proceso de votaciones con la mínima fricción posible y la intención de poder competir con procesos tradicionales de votación.

La solución se ha desarrollado combinando un conjunto de infraestructuras tecnológicas *peer-to-peer* que han permitido implementar todas las funcionalidades necesarias involucradas en la toma de decisiones colectiva. Al mismo tiempo, el sistema es simple y fácil de usar, con una interfaz de usuario intuitiva que elimina las barreras de conocimiento de los usuarios, para facilitar el uso de estas tecnologías revolucionarias.

Acknowledgements

First of all, I would like to thank Jordi Pérez González for joining me on this trip to create a fairer democratic system and to prove that our dream could one day be a reality, one where people can have an impact on the decisions taken by their governments.

I would like to also thank the advisor Jose Luis Muñoz Tapia, as it has allowed us to work fully autonomously, giving us freedom when it comes to tackling the problem we are trying to solve. This has really enhanced our imagination and work capabilities when it comes to facing the greatest challenge that we have faced so far in our engineering careers.

Revision History & Approval Record

Revision	Date	Purpose
0	11/06/2022	Document creation
1	21/06/2022	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Oriol Rodríguez Setó	oriol.rodriguez.seto@estudiantat.upc.edu
Jose Luis Muñoz Tapia	jose.luis.munoz@upc.edu

Written by:		Reviewed and approved by:	
Date	21/06/2022	Date	21/06/2022
Name	Oriol Rodríguez Setó	Name	Jose Luis Muñoz Tapia
Position	Project Author	Position	Project Supervisor

Table of Contents

Abstract 1
Resum 2
Resumen 3
Acknowledgements 4
Revision History & Approval Record 5
Table of Contents 6
List of Figures 8
List of Tables: 9
1. Introduction 10
1.1. Statement of Purpose 10
1.2. Projects Goals 10
1.3. Requirements & Specifications 11
1.4. Methods & Procedures 12
1.5. Work Plan 12
1.6. Incidences 13
2. State of the Art of the Technology 14
2.1. Blockchain 14
2.1.1. Proof of Work 15
2.1.2. Proof of Stake 16
2.1.3. Ethereum 16
2.1.3.1. Ethereum Virtual Machine 18
2.1.3.2. Smart Contracts and Proxies 19
2.2. InterPlanetary File System 21
2.3. Hash Functions 25
2.4. Merkle Trees 25
3. Project Development: 27
3.1. Implementation Tools 27
3.2. Implementation Background 28
3.3. Architecture 30
3.4. Workflow 33
3.4.1. Connection Process 33
3.4.2. Log In Process 33

3.4.3. Registration Process 35
3.4.4. Transfer Process 36
3.4.5. Vote Delegation Process 37
3.4.6. Admin Log In Process 38
3.4.7. Admin Add User to Census Process 39
3.4.8. Admin Remove User from Census Process 40
3.4.9. New Proposal Submission Process 41
3.4.10. Vote Submission Process 42
4. Project Results 44
4.1. File Storage Usage Results 44
5. Budget 45
6. Conclusions & Future Development 47
Bibliography 50
Appendices 52
Glossary 55

List of Figures

Figure 1.1. System Architecture Design and Implementation Tools 12
Figure 1.2. Work Plan.. 12
Figure 2.1. Simple Blockchain Structure... 14
Figure 2.2. UTXO Model vs Account Model 17
Figure 2.3. Types of Proxies 20
Figure 2.4. IPFS Object.. 22
Figure 2.5. IPFS Object Linking 22
Figure 2.6. IPFS Object Commits 23
Figure 2.7. IPFS Nodes State Before Data Fetch 24
Figure 2.8. IPFS Data Fetch Flow Diagram... 24
Figure 2.9. IPFS Nodes State After Data Fetch 25
Figure 2.10. Merkle Tree Structure 26
Figure 3.1. Identification System Architecture 31
Figure 3.2. Governance System Architecture.. 32
Figure 3.3. Flow Diagrams Legend... 33
Figure 3.4. Log In Process Flow Diagram 34
Figure 3.5. Registration Process Flow Diagram 36
Figure 3.6. Transfer Process Flow Diagram 37
Figure 3.7. Vote Delegation Flow Diagram.. 38
Figure 3.8. Admin Log In Flow Diagram 39
Figure 3.9. Admin Add User to Census Flow Diagram.. 40
Figure 3.10. Admin Remove User from Census Flow Diagram 41
Figure 3.11. New Proposal Submission Flow Diagram 42
Figure 3.12. Vote Submission Flow Diagram 43
Figure 4.1. IPFS Storage Analysis Results.. 44
Figure 5.1.Web hosting pricing 45
Figure 5.2. Infura Ethereum services pricing... 45
Figure 5.3. Infura IPFS services pricing.. 45
Figure 5.4. Smart contract gas fees.. 46

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

telecos BCN

List of Tables:

Table 2.1. Ethereum Opcodes Table 18

1. Introduction

This first chapter includes the purpose, objectives, requirements, specifications, and the implementation plan outcome.

1.1. Statement of Purpose

The motivation behind the development of this project is a strong disbelief that we have towards the fairness and democratic representation of the current voting systems.

We believe that the process used in most developed countries to make decisions as a community totally misrepresent the general will of the community, as the decision-making process is not done by the community but by someone elected who is intended to represent this general will.

One of the major drawbacks on allowing a community to directly take part on the collective decision-making process is the elevated cost and infrastructure requirements needed to cast the votes physically in a secure and trusting environment, making it not possible to do it on a regular basis.

Thanks to recent technological developments, more precisely decentralized and trustless networks, also known as peer-to-peer networks, it is now more feasible to implement a system that meets the requirements to host these collective decision-making processes without the need of deploying physical points to cast votes, removing the elevated costs associated with it.

This project is a first step towards a future with peer-to-peer voting systems, a future we believe will be more fair, inclusive, secure, and trustworthy.

1.2. Projects Goals

The defined project goals are the following:

1. Design and implementation of a uniquely identification system for individuals.
2. Design and implementation of a governance system for the identified individuals in the identification system.
3. Both systems must be peer-to-peer and compatible between them.

1.3. Requirements & Specifications

The requirements of the project have been divided following the two systems the project is made of.

Requirements of the identification system:

1. Registration of users must provide a unique identification.
2. Registration of users must be allowed if the user belongs to the defined system census.
3. Registration of users must be protected from attackers with the knowledge of a secret.
4. Registration of users and system census must be publicly accessible and of a peer-to-peer nature.
5. The issued identifications must be transferable between wallets on the network.
6. An administrator must be able to add and remove users from the system census, and the consequences of this modifications must be applied at once.
7. The system must be upgradable without requiring heavy infrastructure modifications.

Requirements of the governance system:

1. The system must be of a peer-to-peer nature and secure.
2. The system must be only accessible to the owners of an identification issued by the identification system.
3. Users of the system must be able to delegate their vote.
4. The system must be protected against a delegation loophole (scenario where all users have their vote delegated, state not possible in normal democratic processes).
5. Any user must be able to create a proposal that it can be voted on.
6. Any user must be able to cast their vote on any proposal being active.
7. Any user must be able to see the full history of proposals of the system.
8. The system must be upgradable without requiring heavy infrastructure modifications.

When it comes to the specifications of the project no division has been made, as both deigned systems rely on similar technologies:

1. User interface of web client nature and implemented with the React framework in combination with Tailwind CSS framework. The programming languages of the implementation are JavaScript, HTML and CSS.
2. System logic on the server side hosted on the Ethereum peer-to-peer network and developed using the Hardhat framework. Communication with the network is provided through an Infura Ethereum node with a RESTful API. The programming language of the implementation is Solidity and OpenZeppelin Solidity contracts model for secure contract writing. For testing purposes, the Hardhat-Waffle and mocha frameworks are used.

3. System storage of large files (the system census) hosted on the IPFS peer-to-peer system. Communication with the IPFS system is provided through an Infura IPFS node with a RESTful API.
4. Communication with the Infura Ethereum API is stablished with the ethers.js library while communication with the Infura IPFS API is stablished with the axios library.
5. Connection with the Metamask wallet is stablished using ether.js from the react application.

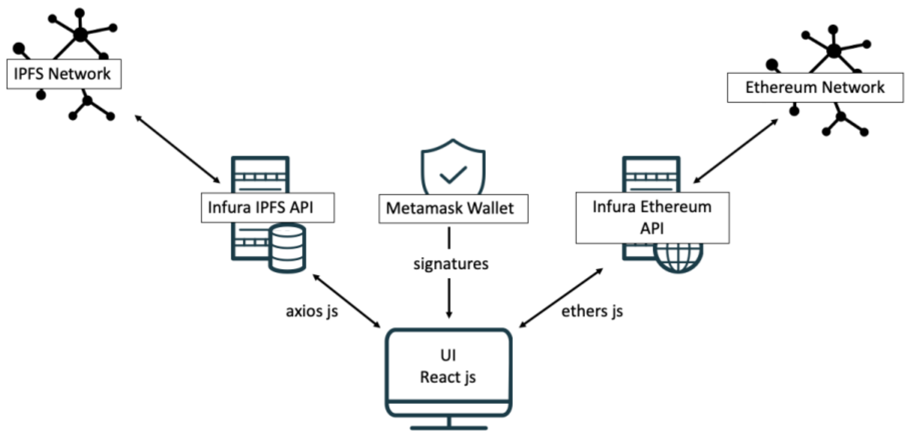


Figure 1.1. System Architecture Design and Implementation Tools

1.4. Methods & Procedures

This project has been designed and developed in collaboration with Jordi Pérez González. In his thesis the complete infrastructure of the system deployed on the Ethereum network is explained in detail, with the technical and philosophical aspects on that part.

So, to fully understand the designed and implemented system, both theses need to be read.

1.5. Work Plan

The final work plan of the project has resulted as follows:

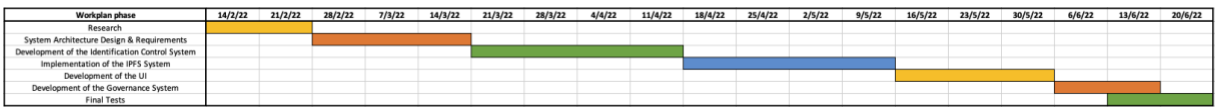


Figure 1.2. Work Plan

1.6. Incidences

Throughout the project many and multiple incidences have been faced, as the project is quite challenging. It is not possible to mention all, but here are some of the most remarkable:

1. The node provider services being down, although this is not recurrent and very odd given the chosen provider, this service downtime moments have resulted in lots of hours wasted between trying to find the origin of the problem and not being able to test out with the peer-to-peer networks. Related to external services, Ethereum Testnets has been used to test out the project on a real blockchain environment, which had led to having problems with the acquisition of ETH tokens to pay for the transactions in these networks, as most of the providers of these tokens have been out of service or under high demand.
2. Data format incompatibilities and conversions have been another of the great issues. The problem has mainly appeared when trying to communicate the web client with the peer-to-peer services, which led to formatting errors and information not being properly decodified on the other end of the communication. This problem has specially accentuated with the Solidity contracts.
3. User interface implementation has been another of the major issues, as it was first intended to be done by a Command Line Interface, so the efforts of the project could focus on the development of the infrastructure, but it turned out to be extremely complicated to use given the complexity of the project and the nature of the wallet providers (Metamask). A complicated decision had to be made, and the problem was solved by implementing a reactive user interface client, which led to big setbacks in the project timing.
4. Proxy and implementation contracts verification procedures have been the last major issue of the project. Verifications on the Ethereum Testnets have been necessary due to security measures practices, testing purposes and academic demonstrations. These verifications have led to some unexpected setbacks on the project timeline as the process to verify contracts implemented with proxies is not trivial and more complex than contracts implemented without them, therefore leading to time consuming operations that were expected to be easy and fast.

2. State of the Art of the Technology

The goal of this section is to provide a general overview of the blockchain technology literature. Introducing technical concepts related to this technology.

2.1. Blockchain

Simply explained, the blockchain technology is a series of blocks linked to each other via cryptographic functions generating a strong and secure chain. Given this technology, it will be possible to record transactions through a secure and verifiable process without any intermediary.

This technology was introduced in 1991 by Stuart Haber and W. Scott Stornetta [1], who first wanted to implement a system where document timestamps could not be tampered with. However, blockchain as it is widely known today, was introduced in 2008 with the introduction of Bitcoin by Satoshi Nakamoto [2], who proposed a blockchain based system that created the first peer-to-peer electronic payments system, a system in which two nodes can send transactions between them without having to rely on an external entity or third party. Thanks to the blockchain technology, a problem that could not be solved for years in different attempts to build a peer-to-peer electronic payments system was solved, the double spending problem.

By design, blockchain aims to be tamper proof. Due to the chaining technique each block links to the previous one with hash pointers. Each block contains a set of transactions, their Merkle representation, a timestamp of its creation, and the cryptographic hash of the previous block. Therefore, the content of block *n+1* has the hash of the block n. Thus, if any actor tries to tamper a transaction in block *n*, its hash will change, and it would instantly compromise the validity of all the following blocks. This way, it results extremely easy to verify the integrity of every node and extremely difficult to alter this chain of blocks that has been created. The structure of the chain is explained in the following figure:

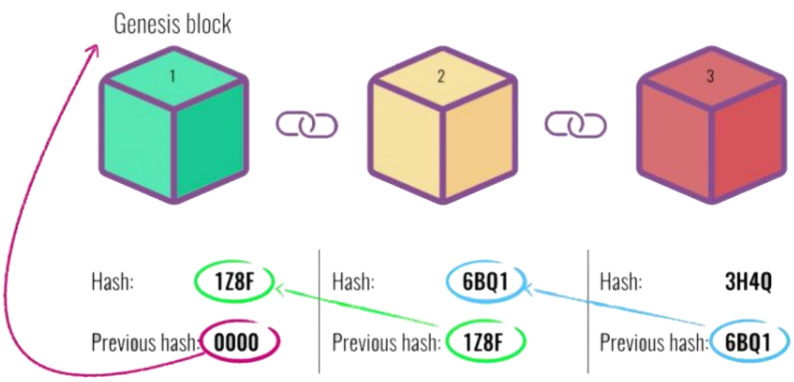


Figure 2.1. Simple Blockchain Structure

Blockchain is based on a Distributed Ledger Technology and its goal is to create a fully decentralised and distributed system, securely storing ledger copies in all the participating nodes. It doesn't rely on a single point of control, which makes the system fairer, more reliable, and considerably more secure. This ensures the integrity of the data, as any attack would require compromising all the copies simultaneously, therefore making it hard to do if the ledger is found to be considerably distributed. If the consensus of most of the network is not achieved the changes would be dismissed.

Blockchain can be classified between public (permissioned) and private (unpermissioned) networks:

- **Public:** Blockchains that are fully decentralized, everyone is free to participate, and is owned by the nodes who take part of the network. Bitcoin and Ethereum are an example of this type of blockchains.
- **Private:** Blockchain that can only be accessed by users with permissions, and its users can only perform specific actions granted to them by the ledge owners. Hyperledger fabric is an example of permissioned blockchains.

Theoretically, a blockchain could be compromised if a bad actor gets to control 51% of the network nodes, known as the 51% attack. By controlling more than half of the nodes, the attacker could verify something to his own benefit, as the blockchain will validate transactions if consensus is reached, as explained before. Blockchain networks leverage consensus protocols to prevent these attacks from happening. Consensus protocols are processes used to achieve agreement on a single data value among distributed process or systems. It enables nodes to achieve a consensus and agree on a sole source of truth, even if some agents fail. Below we will describe the most popular consensus algorithms, the Proof of Work and Proof of Stake mechanisms.

2.1.1. Proof of Work

Proof of work has been the first consensus algorithm used in blockchain networks. The term was first mentioned and formalized in a paper by Markus Jakobsson and Ari Juels in 1999 [3]. Even though it is used in various blockchains, it is primarily known for being the consensus algorithm used in Bitcoin.

In Proof of Work, nodes participating in the network, called miners, must solve a complex mathematical problem to add a block and to earn a reward. This mathematical problem, also named as cryptographic puzzles, consists in calculating a target hash, which is expensive to compute but effortless to verify, making extremely easy to validate previous mined blocks. Hash functions are used as are preimage resistance, meaning that given a hash *h* there is nonother way than brute force to find and input that gives the desired output.

The protocol sets out a difficulty on the cryptographic puzzle to keep the mining time around the target time, regardless of the number of miners and its power computation, to keep the time between blocks the same. For instance, Bitcoin protocol sets the difficulty level by defining a hash starting with a fixed number of zeroes.

When looking for the hash, miners must take as input the hash of the parent block, the transactions contained in the block, waiting for verification, and the timestamp. Then, a random number, called the nonce, must be added to find the targeting hash.

This protocol makes very hard for bad agents to tamper the chain. It would require an enormous computational effort for the attacker, as he would need more computational power than the required to create all the blocks back from the first one he is trying to modify, because it would be needed to recompute all the hashes from all the previous blocks, therefore the blockchain is secured by the computational cost it requires to build it. At the same time, this kind of design makes an attack to the network extremely hard to do in terms of resources and financially unprofitable.

2.1.2. Proof of Stake

When it comes to Proof of Stake, the concept was first conceptualised and proved in a research paper published by Aggelos Kiayias, Alexander Russell, Bernardo David, Roman Oliynykov [4].

Proof of Stake depends on a validator's economic stake in the network. In these systems, there's no concept of miners nor massive energy consumption to compute a mathematical problem. In Proof of Stake, there are validators, who take turns to propose and vote on the next block, and the voting weight of the participants depends on the number of coins deposited.

Everyone who holds the native token of the blockchain can participate in the validation process and proposing new blocks, by locking up their tokens into a deposit. When a block is going to be mined, a signer is randomly selected from the validators based on the amount of stake deposited. If a validator has a faulty behaviour, it will receive a penalization on the amount he has staked, process known as slashing. This way, as in Proof of Stake, acting honestly is more profitable than acting dishonestly, so validators are incentivised to do the correct thing. As mentioned above, the more amount staked a validator has, the higher chances it will have to be selected to validate the next block, so in other words, the more exposure to the downside you have to bad behaviour, the more responsibility you are given.

Proof of Stakes covers some drawbacks from Proof of Work protocol, that caused big blockchains such as Ethereum to initiate a transition to this protocol. One of the main reasons is due to the exhaustive amount of energy needed to run Proof of work systems.

2.1.3. Ethereum

Ethereum is the second most popular blockchain platform. It was founded by Vitalik Buterin, Gavin Wood and Joseph Lubin [5].

When compared with Bitcoin, one of the main differences is that Bitcoin is a transaction optimized blockchain, which means it is mainly designed for the transfer of assets, whereas Ethereum is a code optimized blockchain, which refers it is designed to execute code. In terms of network's state, Bitcoin's is the collection of all coin that have been mined but not yet spent (technically UTXO or "unspent transaction outputs"). On the other hand, in Ethereum the state is made up of objects called "accounts", each account contains its current balance, the contract code if present, the account storage and a nonce, as for number used only once, a counter to make sure a transaction is processed only once.

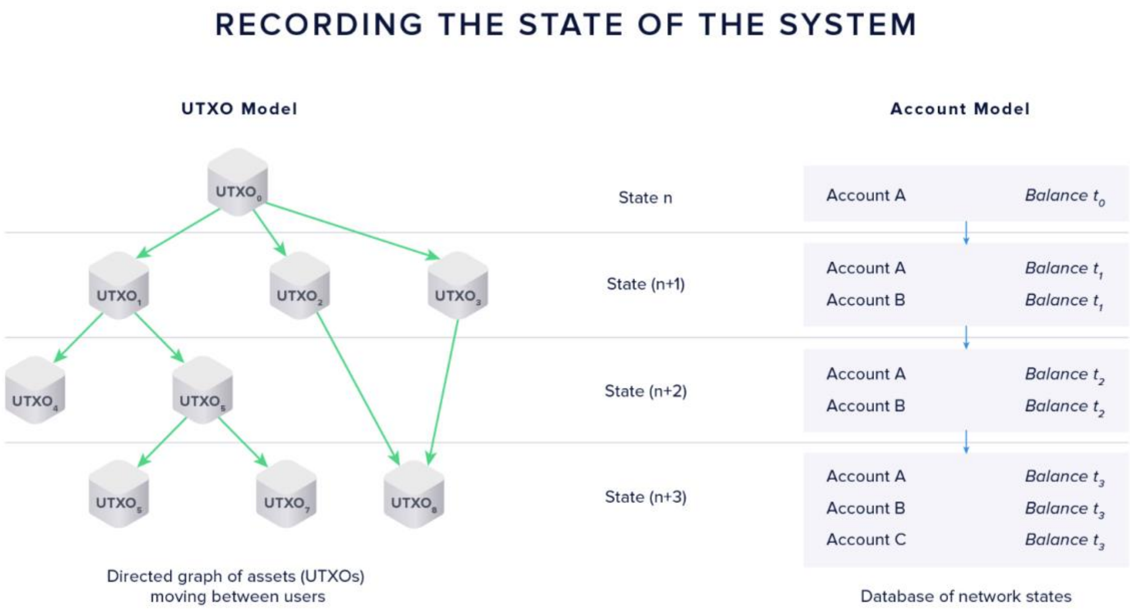


Figure 2.2. UTXO Model vs Account Model

The purpose of Ethereum is to create a platform that enables and facilitates to create decentralized applications with the introduction of smart contracts. At the same time, it serves as a global computing platform where applications run.

Ethereum is fully decentralized, meaning no person owns or controls the protocol. In typical centralized organizations, the board of directors or the executive team have ownership of the decision-making process. In decentralized organizations no one has the final say. Therefore, Ethereum runs a governance process to make decisions about past and future changes on the platform, where all the stakeholders from the community can get involved and have a say.

Ethereum implements an off-chain governance system, where stakeholders can submit a proposal called Ethereum Improvement Proposal (EIP) on GitHub. After an EIP is submitted it goes through a cycle of technical reviews, research, and discussion.

The protocol has its own digital currency called Ether. The main functionality of Ether is to power the network by making users pay for transactions fees. Every time a user or a Dapp changes the state of the blockchain a fee must be paid, this fee is called gas.

The price of gas depends on how much computing power is necessary to apply the changes requested to the network state, and each specific operation in the Ethereum network has a fixed price. This way, users are encouraged to create well-formed transactions and bad actors are disused from saturating the network. This system also acts as a protection to denial-of-service attacks (DDoS).

Gas has no value without the concept of gas price. This parameter indicates the price of each gas unit. Even though each operation has a fixed fee, the gas of price is fluctuant, and can be calculated with the following formula:

Gas Price = Gas Units (limit) x (Base fee + Tip)

Base fee is the smallest price per unit of gas a user must pay if he wants his transaction to be included in the block. Ethereum calculates this base fee based on the demand for block space at any time. In addition, users are also expected to include a tip, which will determine how quickly a transaction will be processed. The result of this operation is given in gwei, which is a denomination of Ether, where 1 Ether = 10^9 gwei.

One of the core problems of Ethereum is the scalability of the platform. The reason of this problem is common to all blockchain, the size of the database underlying the blockchain grows without bounds, in proportion to the number of users. For this reason, Ethereum is planning to be upgraded to Ethereum 2.0, with two majors' changes:

- **Proof of Stake**: Ethereum is transitioning from Pow to PoS. The reason of this change is because the latter is less energy-intensive and would allow to increase the process capacity of the network.
- **Sharding**: This process splits the blockchain into multiple blockchain known as shards that can be stored in different nodes. This would allow validators to handle a smart part of the workload, increasing the transactions per second rate.

2.1.3.1. Ethereum Virtual Machine

Ethereum Virtual Machine (EVM) serves, as the name suggests, as a virtual machine for the Ethereum's' network, which essentially creates a level of abstraction between the executing code and the executing machine. It acts like a decentralized engine maintained by thousands of connected computers running an Ethereum client. Every Ethereum node runs on the EVM to keep consensus across the blockchain.

EVM uses a set of instructions to execute specific tasks, called opcodes. When a contract is compiled, the EVM transforms the code into opcodes, which allows the machine to execute almost anything. To efficiently store opcodes, they are encoded to bytecode. Every opcode has an allocated byte and has associated a gas cost. Developers must consider this when coding to optimise the contract and make the interaction with the contract cheaper.

Opcode	Name	Description	Gas
0x01	ADD	Addition operation	3
0x20	KECCAK256	Compute Keccak-256 hash	30 + 6 * (size of input words)
0x54	SLOAD	Load word from storage	800
0x55	SSTORE	Store value to storage to non-zero from zero / from non-zero to non-zero	20.000 / 5.000

Table 2.1. Ethereum Opcodes Table

The gas cost of four of the EVM's primary opcodes is described in the table above, where each word is 32 bytes. As it can be seen, storing data in the blockchain is one of the most expensive operations. This was designed because once some data is written in the blockchain this data is going to be stored in the blockchain history if the network keeps running, ideally, forever. This way, the network tries to disincentivize user from taking valuable storage resources, and this also explains that the gas price of the SSTORE opcode cost 20000 gas when you set a value from zero value to non-zero, but only 5000 gas when changing the data value from non-zero to non-zero.

This design makes sure that the Ethereum active nodes (nodes that store all the historical data of the network since the genesis block) are incentivised to keep upgrading and maintaining their hardware to store all this data, while the network users are incentivised to optimise as much as possible these resources.

2.1.3.2. Smart Contracts and Proxies

One of the pillars of the Ethereum network are the smart contracts. It's essentially a program that runs on the Ethereum blockchain and consists of a set of instructions (code) and data (state) stored at a specific address. As a result, smart contracts are a kind of Ethereum accounts, with a balance and the ability to transmit transactions throughout the network.

The smart contract is not controlled by the user; instead, it is deployed to the network and executed as a program. By sending transactions, user accounts can engage with the smart contract's capabilities. Smart contracts, like conventional contracts, can define rules that are enforced by the code that sits within them. Consider it like a vending machine: you can interact with it, but to acquire what you want, you must meet certain criteria; otherwise, you will not receive it.

Smart contracts inherit the properties described by the blockchain since they are allocated on it. Because of this quality, smart contracts are a piece of software that does not require trust; if two parties agree on a contract, they may rest assured that the contract will be executed as they both agreed.

However, a fundamental problem with immutable contracts is that are immutable. Bugs and security vulnerabilities are inadvertently written into permanent contracts that could make them completely worthless. Contracts that are immutable cannot be upgraded or fixed, resulting in contracts that become steadily inferior over time.

To make contracts upgradeable, a proxy pattern must be used. The underlying idea of using proxies is simple: instead of interacting with the implementation of the smart contract directly, users interact with a proxy that can directly call or delegate the implementation contract. Proxies can be classified in different types; the following figure gives a general vision of these types:

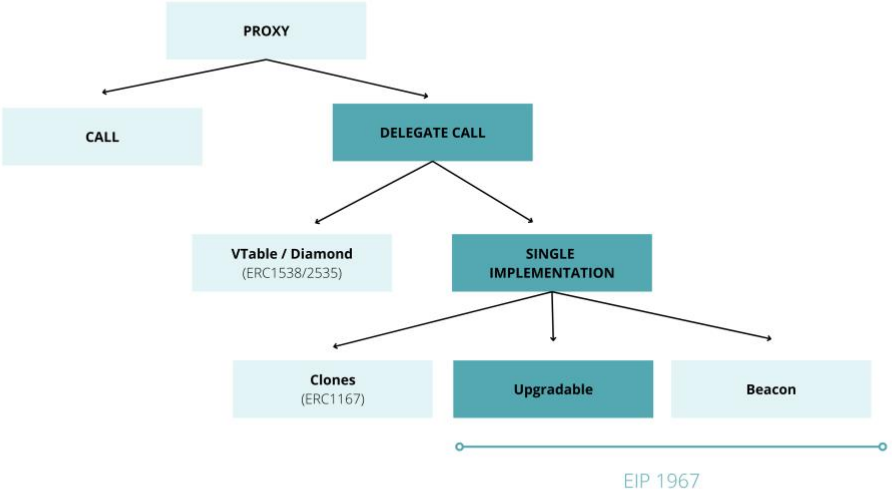


Figure 2.3. Types of Proxies

On the first level we can differentiate between proxies that delegates call to the implementation contract and proxies that directly calls the logic contract. The difference between these two is that the latter directly calls the implementation contract, which means that the storage used is the end contract, therefore, your proxy is purely a router. Whereas, when using delegate call type proxies, the state is saved on the proxy contract, with these proxies is where upgradeability works, as no migration of the storage is needed.

There are two implementations of delegate call proxies. First one, VTable / Diamond, makes use of the diamond pattern and delegates the calls to different modules depending on the function signature. On the other hand, single implementation proxies delegate all the calls that are incoming to a single implementation.

Single implementation proxies have three distinct types.

The first one, clones, is a type of proxy wherein the address that is being delegated to is in the immutable storage, meaning that is part of the proxy's code compiled on the bytecode.

Secondly, there is the proxies specified on EIP1967, which store the address of the implementation in storage, this way, it can be updated. This type of proxies is either upgradeable or beacon type. The idea behind the beacon contract is re-usability. If several proxies are pointing at the same logic contract address, then, would be needed to update all proxies every time the logic contract is upgraded. As this can become gas intensive, beacon contract updates the address of the logic contract in all the proxies that are pointing to this logic contract.

The EIP 1967, where these last two proxies are introduced, is mostly about where in the storage space of the contract should the address of the implementation and its related values be stored.

One important thing when using this type of proxies, is the consistency of the storage layout through the upgrades. When upgrading contracts, it is possible to add more variables, but you cannot remove them nor change the order of declaration. This is because every variable defined, is given a storage slot, so first variable defined is given the slot zero and so on with the other variables, basically it is the linearized version of the contract who figures out which slots are used for what.

Therefore, the proposed EIP defines three reserved storage layouts, one for the logic contract address, one for the beacon contract address (if it is used) and the last one for the admin address (if implemented). It also defines three events, emitted when one of this storage gets updated.

2.2. InterPlanetary File System

The InterPlanetary File System, abbreviated as IPFS, is a file system with peer-to-peer distributed characteristics. This system appears as a solution to the problem that most of the Internet file storage services are centralized and therefore have a single point of failure. The IPFS is set to solve this file storage centralization problem while keeping the characteristics this services offer, high data throughput, low latency and affordable costs.

To ensure a functional peer-to-peer file storage system, the most important aspect is to guarantee file data immutability. This is achieved by ensuring the verifiability and integrity of the data as well as the setup of an environment where no trust is needed.

Data integrity is necessary to ensure that the data you ask for is the data you got. This is achieved creating an identifier from the data itself with the use of hashing and cryptography techniques. This implementation leads us to the data verifiability, being obtained with a new addressing system.

Traditional Internet services are based on location addressing, which means the content can change but the uniform resource locator (URL) won't. This can lead to data being mutated and users not being able to verify the data integrity of the requested file. With IPFS, this problem is addressed with content addressing, which means that the address associated with the content depends on the content itself, so it is verifiable that the content has not changed. This is achieved by making use of hashing techniques.

This content addressing technology also allows that if multiple files containing the same information are uploaded to the system, only one will be stored, as they will share the same hash, making the network more efficient.

About the infrastructure, the IPFS peer-to-peer file system architecture is based on nodes, which make the base foundation of the system. Without nodes running on the IPFS network, the network does not work. This design is what has made possible to have a decentralised, censorship resistant and immutable data storage system, but at the same time it introduces some major drawbacks that will be mentioned later.

The system works as that every node on the network keeps a cache of the files it has downloaded and responds to requests from other nodes that want any of the files that they have in their cache. This gives to the system a major speed up if the files being used are hosted in the nearby community, so files do not need to be retrieved from thousands of miles away, also making the Internet more efficient. At the same time the system presents worse loading speeds and less efficiency when files are hosted far away and in multiple nodes.

To ensure a permanency of the file being shared in the IPFS network, a file can be pinned in a node, which will make the node store that file on its cache until it is unpinned.

It is also worth mentioning that some file-sharing services offer permanent sharing of files on the Internet through IPFS with this file pin feature. So as a customer you get guaranteed that your file will be always available and stored, at least in the service provider nodes, if it is not unpinned.

Simple Technical Overview

File storage is implemented with IPFS objects, which have a data parameter, where information can be stored up to 256k bytes of data, and a links parameter, which links to other IPFS objects.

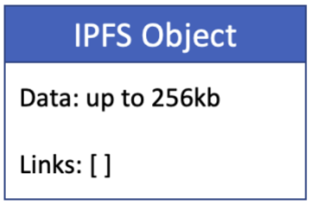


Figure 2.4. IPFS Object

The latter parameter allows to store files bigger than 256k bytes. This works by creating a IPFS object with an empty data parameter that links to all the IPFS objects where the file has been stored divided in data chunks that can be of up to 256k bytes.

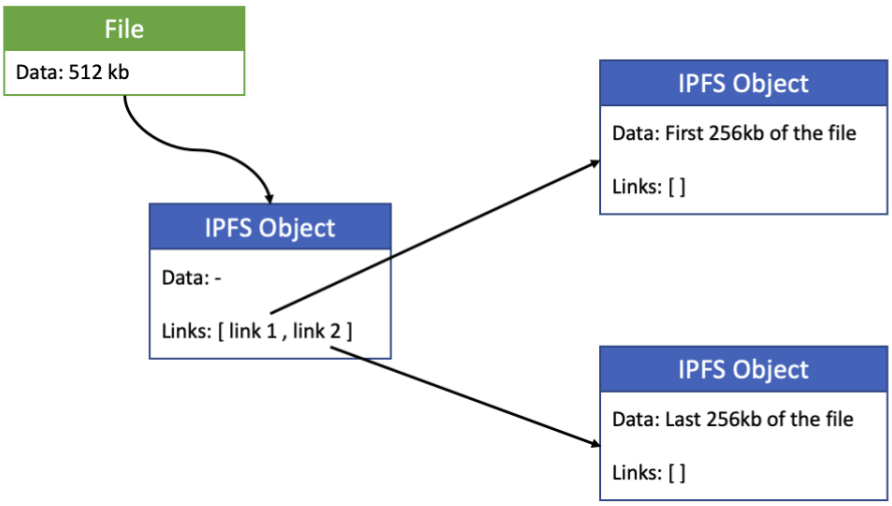


Figure 2.5. IPFS Object Linking

IPFS also supports files versioning. This functionality works by creating commit objects which link IPFS objects between their corresponding versions. These commit objects are created any time an updated version of a file is uploaded to the network, therefore linking this latest version with its earlier version through the parent parameter. This functionality makes it possible for the whole versioning of a file to be accessible to other nodes on the network.

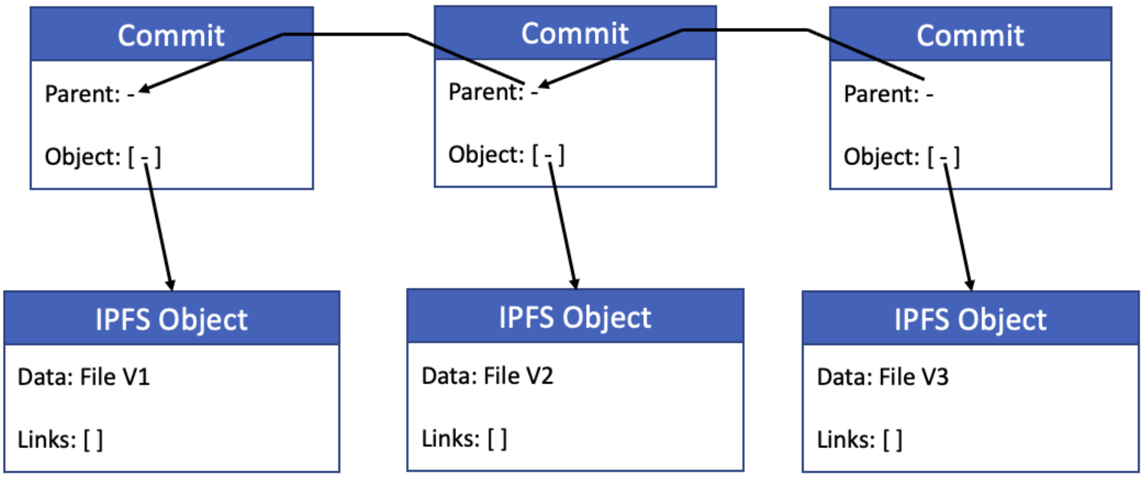


Figure 2.6. IPFS Object Commits

The hash algorithm used by default in IPFS is the SHA-256 algorithm, although other algorithms are also supported. This hashing algorithms create what are called the content identifier (CID), which as explained before, it is what constitutes the content addressing system.

Currently two versions of the CID exist:

1. Version 0 (V0):

Based on base 58-encoded multi-hashes, this is the first version of the CIDs. This design implies more simplicity but at the same time less flexibility. Version 0 CIDs can be identified as 46 characters starting with the "Qm" characters.

2. Version 1 (V1):

In version 1 CIDs extra identifiers have been introduced to give more clarity on which representation is used, along with the content-hash itself. These identifiers are the following:

a. Multibase which specifies encoding used in the remainder of the CID.

b. Version identifier.

c. Multicodec, which specifies the format of the content.

Soon the system will completely migrate to only using V1 CIDs.

Data is structured in the system combining this content identifiers with directed acyclic graphs (DAGs). This DAGs are implemented in the IPFS system as Merkle DAGs, a structure where each node is uniquely identified by the hash of the content it holds. Is like using this defined CIDs but for data layer architectures design and optimization. This allows to create files and directories structures using the content addressing system, which optimizes the data storage resources.

When it comes to the implementation of the discovery function, so the function that finds what peer is hosting the content a user is asking for, the system has been implemented with distributed hash tables (DHTs). A distributed hash table is a table composed of key-value pairs; therefore, a distributed hash table is split across the peers of a network. The peers store a routing table, where they keep a cache of the routes to some of the content. So, for a user to get the content it wants it needs to ask these network peers for the requested content. This peer-to-peer communication is implemented with the libp2p project.

The following figures gives a good overview on how the process of fetching data works on IPFS.

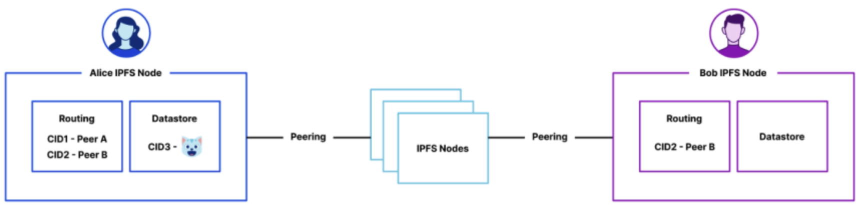


Figure 2.7. IPFS Nodes State Before Data Fetch

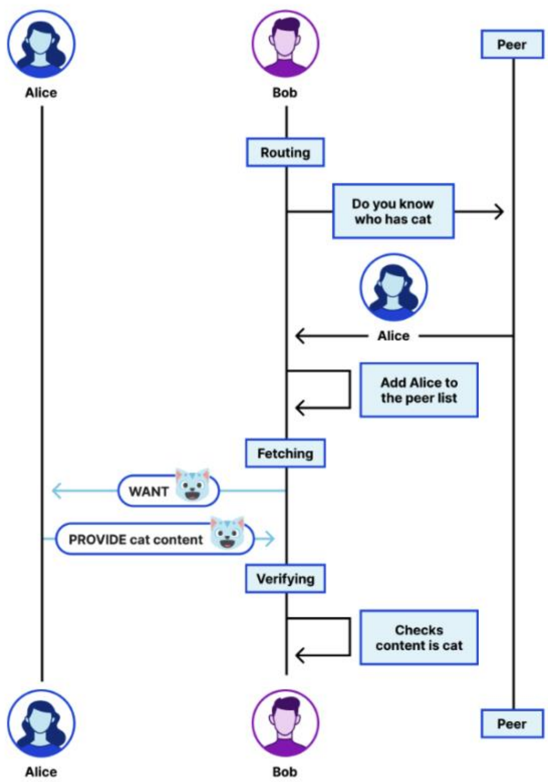


Figure 2.8. IPFS Data Fetch Flow Diagram

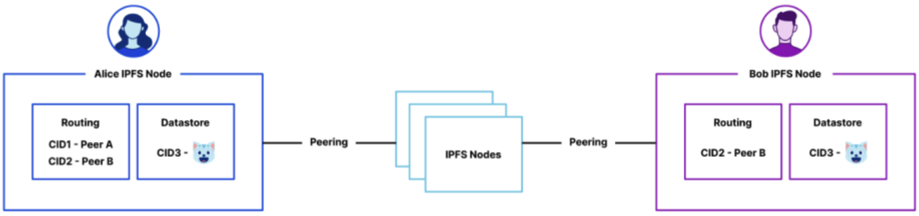


Figure 2.9. IPFS Nodes State After Data Fetch

2.3. Hash Functions

Hash functions are a cryptographic function which generates a fixed-length result from a given output.

A secure hash function must satisfy the following criteria:

1. **Deterministic:** the same input always creates the same output.
2. **Collision resistance:** meaning that no two different inputs generate the same output.
3. **Pre-image-resistance** given a hash *h* should be hard to find any input such that its output equals the hash *h.*
4. **Second preimage resistance:** given an input x_1, it should be infeasible to find another input x_2 such that hash(x_1) = hash(x_2).

2.4. Merkle Trees

Merkle trees are a type of binary tree where every node on the tree is represented by a value that is the result of cryptographic hash function. The tree is created by repeatedly hashing pairs of nodes from left-to-right until there is only one hash left, this hash is called the tree root hash.

Merkle trees allows us to efficiently verify data. It lets one party prove that a particular data item is in the given data structure to anyone who knowns the tree root hash by presenting a proof. Proofs are the shortest list of additional nodes in the tree needed to compute the tree root hash and are computationally easy and fast. So, to build a verification system for a given data base, it is only needed to store the tree root hash, which is extremely more efficient that storing the whole data base.

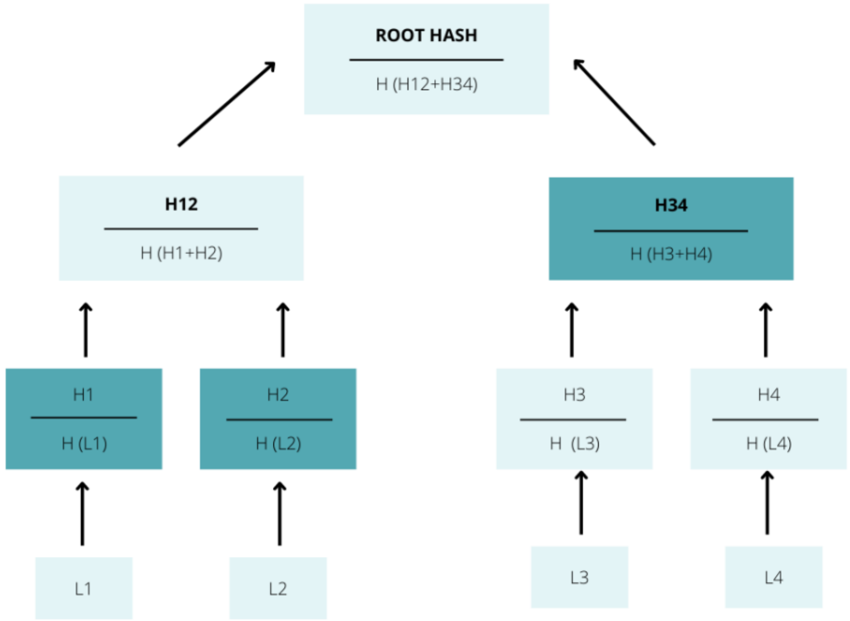


Figure 2.10. Merkle Tree Structure

Figure above shows the verification process for Merkle Tree. For a given set of data blocks, *D = {L1, L2, L3, L4}*, only the hash root is needed to verify the complete set. However, to prove that a data block is contained in the tree, for example L1, the proof generated would be [H(2), H(34)]. Providing these two nodes the root hash can be reconstructed, proving that the data block is contained in the tree.

In blockchain systems, Merkle trees are an important scalability feature, all transactions are stored in the root hash of a Merkle tree. This structure provides an efficient and effortless way to verify that a transaction has been added to a block and provides a way to check if the transactions have been tampered, as a slight change in a transaction would lead to a completely different Merkle tree root hash.

3. Project Development:

This chapter includes the methodology of the project development. No code will be included, but during the explanations, references on where to find the implementations will be added.

All the project code can be found on the GitHub repository link found in the appendices section.

3.1. Implementation Tools

- React JS

The JavaScript framework used to develop the user interface. It allows to implement interactive UIs that change depending on the applications state. It works with a component-context design.

Components, more precisely, function components, serve the same purpose as JavaScript functions but work independently and return HTML code. This allows each component to manage their own state. On the other hand, contexts let us handle the logic and control the data and states shared between components, making it simpler and more structured to create complex UIs. Contexts is a great improvement on UI development as it allows us to share states without having to use the classical React props. The logic is written in JavaScript.

When it comes to the UI states, React offers us the React hook states, which allows to implement those without having to write a class. This has been the approach used in the project implementation.

- Tailwind CSS

Tailwind CSS is a CSS framework that allows to quickly create bespoke user interfaces. It's highly configurable, providing the necessary functionalities for the developer to create personalized designs in a very simple modular way.

Tailwind has been used to implement all the CSS in the React application.

- Vite JS

Vite, as stated by the developer team, it a rapid development tool for modern web projects. In other words, it allows to quickly set up the environment required for the web application without most of the requirements and steps that would be needed using other methods.

It has been used as the web client app setup.

- Ethers JS

Ethers is a JavaScript library used to interact with the Ethereum blockchain. It handles all the functionalities required for interaction with wallet providers and Ethereum node services providers.

- MerkleTree JS

MerkleTree js has been the library used to implement all the functionalities to compute the necessary Merkle tree leaves, roots, and proofs.

- Axios JS

Axios is a promised-based HTTP client for JavaScript. It has been used as it allows to handle HTTP requests from the browser, a functionality necessary to interact with the IPFS Infura API.

- Infura

Infura has been the infrastructure provider to connect to the peer-to-peer networks used in the project, basically IPFS and Ethereum. It works via API, which is then connected to a node from the respective network.

- Etherscan

Etherscan is an Ethereum network explorer. It provides all the information to see what is happening behind in the peer-to-peer network. In this project it has only been used to verify the deployed contracts.

- Visual Studio Code

Visual Studio Code has been the used IDE to develop the whole project.

3.2. Implementation Background

To understand the explanation of the project development, some previous definitions about the identification system need to be made:

1. ID: stands for the sequence of 8 numbers followed by a capital letter that define the unique identifiers of the system.

2. ERC721 ID: the instance of a system ID inside the Ethereum blockchain. Implemented using the ERC721 standard, also known as non-fungible tokens. What uniquely identifies them, the token ID, is the decimal ID of the designed system, a format variation of the ID later explained.

Also related with the identification system, 3 different types of ID have been defined. These definitions have been done for data communication purposes between the Ethereum and IPFS networks with the web application client. The definitions are as follows:

1. Decimal ID: stands for the numeric format of the ID. Composed by the sequence of 8 numbers that form the ID followed by the ASCII number of the capital number, resulting on a 10 number sequence. This implementation has been done due to Solidity (language used to program the Ethereum smart contracts) only accepts unsigned integers of 256 bits as token IDs for the ERC721.

2. Address ID: stands for the formatted version of the decimal ID into a Solidity address variable. This implementation is used for the governance system, as it works with addresses. The formatting has been implemented by computing the 256 bits hash function to the decimal ID, then extracting the last 20 bytes of the resulting hash.

3. Census ID: stands for an address concatenated with the decimal ID. Implemented for the registration access, which are address-ID pairs.

For the implementation of the web client, some tools have been developed. They can be found in the Tools package. Their definitions will be done here, for better understanding later:

1. Cryptography Tools

In cryptography tools all the Merkle tree related functionalities are implemented. It is basically made up of 4 functions, and all 4 compute a different part necessary to operate with the Merkle tree.

The implemented computations are the Merkle tree leaf, root, proof and the complete Merkle tree.

The Merkle tree leaf is computed from a census ID, while the rest are computed from complete census, which is made of all the census IDs of the system.
Libraries used for this implementation are MerkleTree js and Ethers js. The latter is only used for the computing of the 256 bits hash function, so the hashes are computed accordingly with the ones in the Ethereum network contracts.

2. Web3 Tools

Web3 tools contains all the methods recurrently used in the different React contexts to interact with the Ethereum blockchain. It has been implemented using the Ethers js library.
The developed functions can be categorized in two types.
The functions used to interact with the blockchain, which are the function that creates the Ethereum connection provider (via Infura), which instances an object that provides the required functionalities, and the function that creates an Ethereum contract object (makes use of the provider instance), which implements the contract functionalities. Finally in this category there is an implemented function to request to the ERC721 contract the active census (as the ERC721 IDs issued). This last function is not as generic as the previous two.
The following category is the functions to check connection status, which are simple functions to check if whether the user has the wallet installed and the connection with the Ethereum node stablished. These functions are used to update the UI state variables.
Web3 tools have been implemented using the Ethers js library along with the Ethereum object instance obtained from the React window global object, which is implemented using the Web3 library.

3. IPFS Tools

IPFS tools implements basic HTTP request functionalities to interact with the Infura IPFS node that gives access to the IPFS peer-to-peer network.

Only contains 3 basic functions:

> 1. Get census function: an HTTP POST that requests the to the IPFS node the census file stored inside the peer-to-peer network.

2. Post census function: an HTTP POST to upload the new census file to the IPFS network. Note that this function requires the Infura project secret authentication, since the Infura API hosts the IPFS service as a project in its API service.
3. Unpin census function: an HTTP POST to unpin the uploaded IPFS file, so the Infura garbage collector can remove its file, keeping file storage usage optimized. Note that this function also requires authorisation.

All functions are implemented using the axios js library.

It must be noted that this tool treats the census file in a CSV like format.

4. Format Tools

Format tools only contains two functions:

1. The format ID function, which converts an ID to decimal ID format and the other way around.

2. The get time function, which returns the time in days, hours, minutes, and seconds between two given blockchain network blocks. The returned time is an approximation at the worst-case scenario. In the Ethereum network, a block takes an average time between 12 and 14 seconds, so the 12 seconds approximation has been taken.

Finally, about the CSS gradient designs, it has been used an online gradient render tool. The tool is called CSSHero, and can be found in the following link: csshero.org/mesher

All CSS templates implemented in the project can be found in the index.css file.

3.3. Architecture

The user interface has been designed to be only an interacting tool with the Ethereum and IPFS networks. It does not implement security nor a log in session system, even though it only shows the different sections and functionalities under certain conditions, a skilled attacker could bypass those, and could even send a transaction to the system infrastructure on the Ethereum and IPFS networks, where it would be rejected if it does not meet the requirements.

About the UI structure, it is based in two different functionalities, depending on the role of the user connected. There is a user, who has the default UI view, and then there is an Administrator, who also has the default UI view plus some extra components. Both will be later explained in detail on the Workflow section. All administrator related functionalities and states have been implemented in a different context, the AdminContext. Only exception is the owner log in state, later explained in detail.

In terms of storage of system state and parameters, all what is necessary to operate within the system is stored in the peer-to-peer networks, either IPFS or Ethereum.

The only parameters that are hardcoded in the web client app are the following:

1. The system contracts addresses, so the web client can locate the contracts in the network to interact with them.

2. The system contract ABIs, that contain the bytecodes of the contracts, necessary for the web client to know what functions and parameters the contract has and how to execute them.

3. The Infura API services addresses, for the web client to locate and interact with them.

The architecture will be broken down into two parts, the identification system, and the governance system. Two diagrams have been provided for an explanation of the systems architecture.

Identification System Architecture

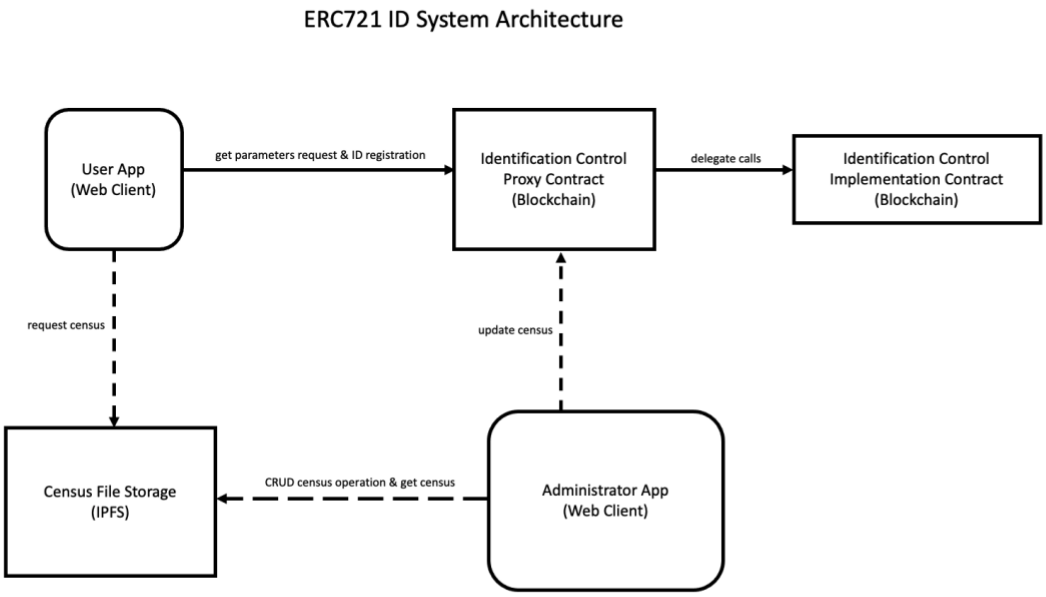


Figure 3.1. Identification System Architecture

As it can be seen, the web client application interacts only with the IPFS network for file storage and with the Ethereum network for identification control functionalities.

Also, as shown in the diagram, the interacting functionalities between the user and the administrator differ.

The system functionalities are therefore divided between the web client and the external peer-to-peer networks.

Governance System Architecture

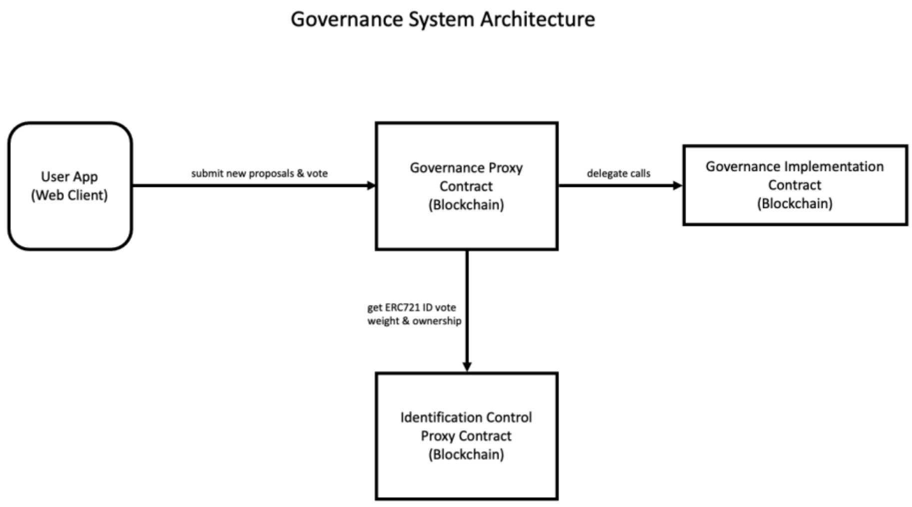


Figure 3.2. Governance System Architecture

As for the governance system architecture, the web client interacts solely with the governance contract that implements all the functionalities. Then it is this governance contract which directly interacts with other contracts during some processes later explained in the workflow section.

There are no different functionalities between an administrator and a user in this part of the system.

About the implemented components and contexts, they are all mentioned in the processes they are involved in, so it can be more clearly understood the context of the implementation decision.

About the proposals process, they have two possible states that affect the UI and its interoperability with them. Active and inactive state.

A proposal is found in an active state while the current Ethereum block has still not reached the marked end block, so a user will be able to submit his vote during this period (being the case that still it hasn't).

An inactive proposal is when it is terminated, therefore no interactions can be done with it.

3.4. Workflow

In this section it will be explained it detail how the different processes have been designed and implemented.

The flow diagrams have the following legend for the specified connections:

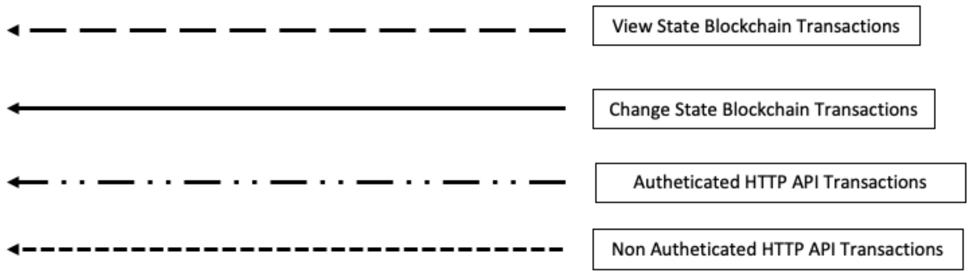


Figure 3.3. Flow Diagrams Legend

3.4.1. Connection Process

This process is the first one of all, as it is the first interaction between the user and the system. When the web client is first loaded, it is shown the welcome screen, where there is some information displayed about the project along with a single button, the Connect Wallet button.

The Connect Wallet button is the only possible interaction the user has with the system when the UI is found in this state. When interaction is initiated with the React button, the connection process is initialised. The web client will request an installed Metamask wallet, and then this wallet extension will request the user to sign the connection to the site, both logic processes implemented with Ethers.js library.

The graphic interface of the project can be found on the LogIn.jsx component while the logic functions in the LogInContext.jsx.

Once the signature is received, the log in process starts. After signing and connection, the connected account is saved in the connectedAccount state hook.

3.4.2. Log In Process

Log in process starts from the moment the user has connected its Metamask to the Ethereum network, as the web client will now be able to talk with the system contracts, where all the states and necessary parameters are stored.

The UI view for the user to proceed with the process is found in the LogIn component. It has a React select component, which will allow the user to choose between the different owned ERC721 IDs, stored in the IDs state variable mentioned in the following paragraph.

Complementing, a submit button is implemented to proceed with the log in process on a valid selection.

About the UI states, this process only updates the IDs state variable and the logInId state variable. The first is an array with all the owned ERC721 IDs. In case the user does not have any remaining ERC721 IDs, the view changes to the registration page, otherwise is kept where it was. The latter contains the state on what ID the user has selected to proceed with the log in, so interactions with the Ethereum contracts will be done with this ID. These states are found in the LogInContext.

The process flow diagram is described below, with all the details and corresponding computations.

A notation must be made, so it is clearly understood how the log in process is implemented. The checks the process goes through are as follows:

1. Check if the connected wallet is the owner of the contract that handles all the identification system logic. If positive, the log in process proceeds differently, into what has been defined as the admin log in process, later explained. If negative, the log in process continues. This check is done by requesting via Ethers the owner parameter of the contract.

2. Second check is the connected wallet ERC721 IDs balance, via the balance of contract function. If positive, therefore some balance bigger than 0 is returned, the log in process proceeds to iterate though this balance to obtain the ID number of each of the ERC721 IDs owned. If negative, log in process terminates, on what would be the registration process, later mentioned.

After the checks, the web client displays what ERC721 IDs the connected account owns.

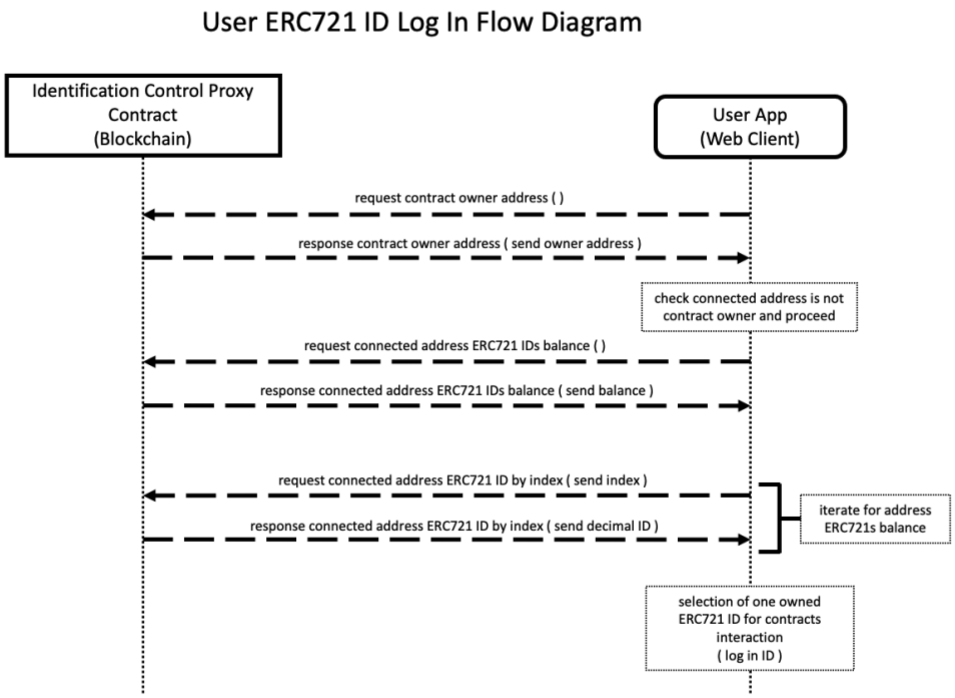


Figure 3.4. Log In Process Flow Diagram

As it can be seen, the log in has not been implemented with a traditional log in session that would require the implementation of an API. Instead, it has been opted to be implemented with the contracts, therefore delegating all security functionalities to the blockchain.

3.4.3. Registration Process

The registration process is the point of entry of users to the system. In this process is where IDs are created and issued to the users. An ID is implemented with an ERC721 contract which creates a non-fungible token. This implementation gives freedom to the user to transfer its identification between different wallets within the Ethereum ecosystem.

To successfully register, a user must have his wallet registered in the census along with the sequence of number and digits an ID is made of. The process in which a wallet and an ID is introduced in the census will be later explained in detail.

On the UI, to have access to the screen where the registration ID is entered, it is required for the user to have his Metamask wallet connected to the network, but to not own any ERC721 IDs. The state of not owning any ERC721 IDs is implemented in the IDs state variable, found in the LogInContext.

When the UI is found in this state, a React text input form is presented to the user, where it must introduce the ID. Once introduced, the form is submission is initialised with a button. The web client then proceeds to check if given ID along with the connected wallet address are registered in the census.

The mentioned interactive components are implemented in the LogIn component.

Web client requests to the contract handling the registration the file hash of the census, which is stored in the contract parameter. With this file hash it can now be retrieved the census file from the IPFS network, where the file is stored. Note that this request is done using content addressing, therefore the hash is what locates the content on the network while at the same time is what allows us to verify the data integrity.

The IPFS requests are done with the IPFS tools implemented and explained before.

The whole purpose of requesting the census is to ensure the ID has been correctly written and it is registered accordingly in the census, so no transaction to the blockchain network is initialised with wrong parameters. By doing this it is avoided the situation where the user would pay the gas fees for the transaction just to be later rejected, as the formal check is later done in the Ethereum contract.

The process flow diagram is described below, with all the details and corresponding computations.

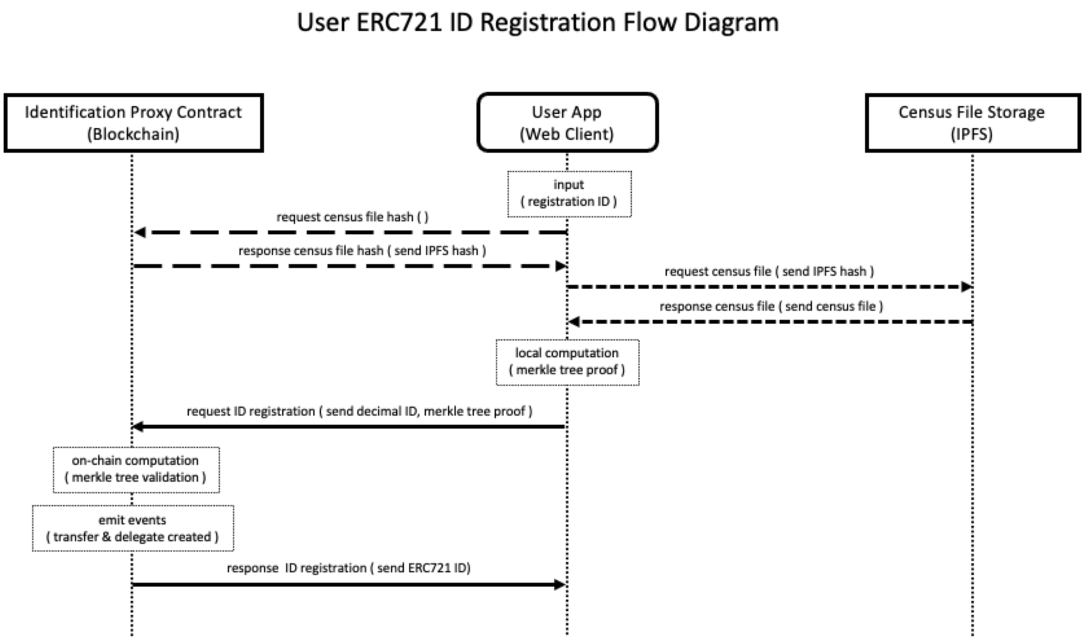


Figure 3.5. Registration Process Flow Diagram

3.4.4. Transfer Process

The transfer process is not a strictly necessary to interact with the designed system. Rather it has been implemented to offer optionality to the user, so a change of wallet can be done, if necessary, even allowing for a wallet to handle multiple ERC721 IDs.

It implements the ERC721 ID transfer functionality. The UI view for the user to execute the process is found in the LogIn component. It has a text input for the user to enter the destination address and a button to initialize the process once the destination address is entered.

About the UI states, this process only updates the IDs state variable, which is an array with all the owned ERC721 IDs. In case the user does not have any remaining ERC721 IDs, the view changes to the registration page, otherwise is kept where it was. This state is found in the LogInContext.

The process flow diagram is described below, with all the details and corresponding computations.

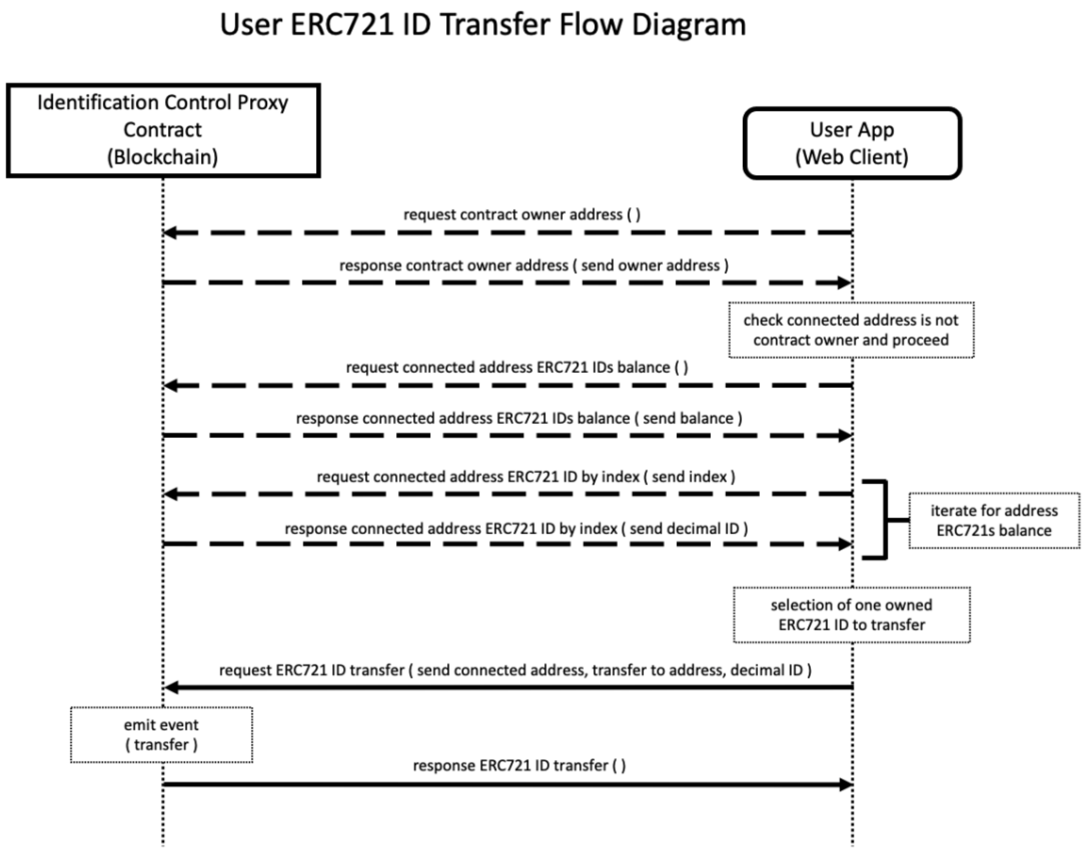


Figure 3.6. Transfer Process Flow Diagram

3.4.5. Vote Delegation Process

The vote delegation process implements the functionality for a user to delegate its vote to another user (identified with an ERC721 ID). This allows the user to let another user vote for him. In other words, the voting power of the user that delegates become 0 as the voting power of the delegatee (user who receives the delegation) is increased by one unit.

The delegate ID is entered with a React text input on the DelegateVote component. It also has an interactive button to initialise the process once the ID is entered.

About the UI states, this process only modifies the delegatee state, which displays in the UI to what ID the user has his vote delegated. State found in the GovernanceContext. This component has been implemented with a view and hide functionality, activated, and deactivated with an interactive button. The view/hide functionality state is implemented in the ViewStateContext. In this case the variable is called viewDV.

The process flow diagram is described below, with all the details and corresponding computations.

Note that in the Ethereum network side, if a user delegates his votes and had some delegation to his ID, these delegations are returned to their respective IDs. The system notifies this via emitted events, so when the delegatee state variable is updated, the UI of this ID that had their vote delegated will also be updated. This is done to avoid a loophole where all users have their vote delegated, which would make the system enter a non-possible state in any decision-making process found on democratic countries.

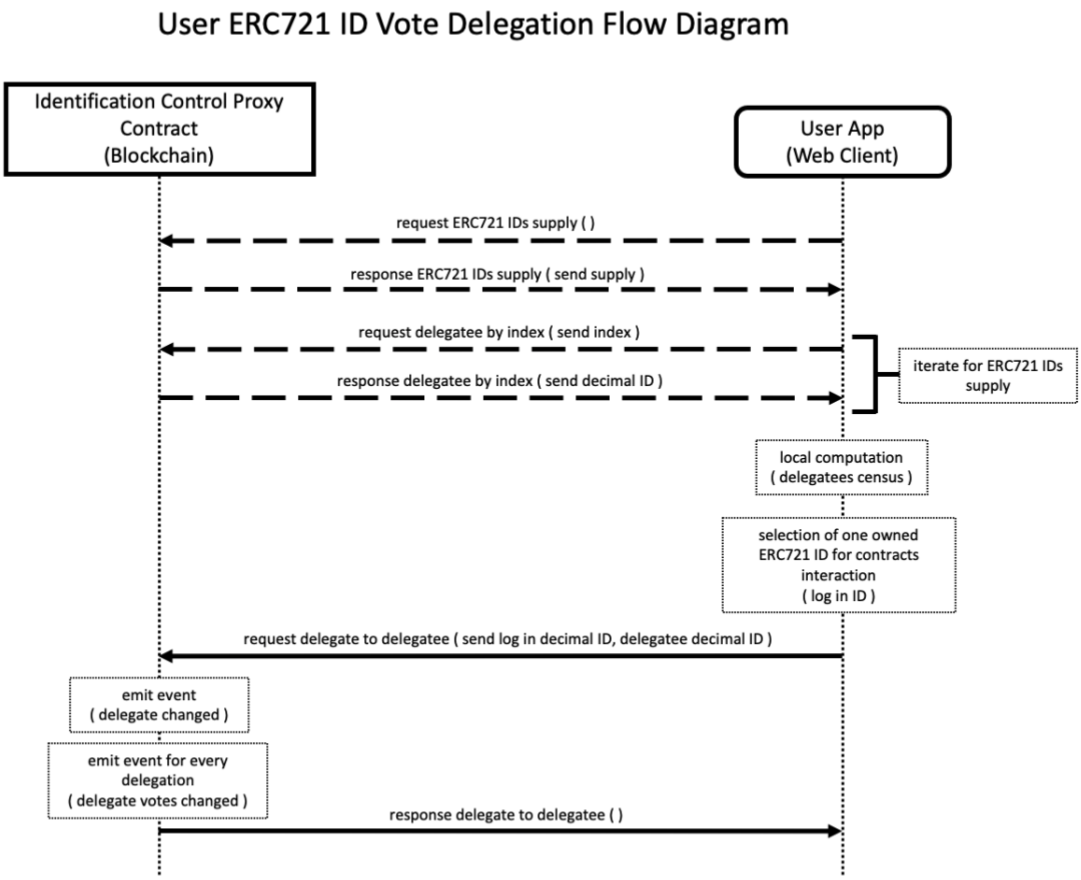


Figure 3.7. Vote Delegation Flow Diagram

3.4.6. Admin Log In Process

The admin log in process has de same functionality as the user log in process, only difference is the two processes end up in different states given the result of the first check. This different termination is what gives access to a user default UI or an admin UI.

To access the admin UI this process is strictly necessary.

It implements the administrator log in functionality. The process is implemented with the user log in process, with the only difference being the React states it modifies.

About the UI states, the process modifies the isOwner and IDs states. The first one, sets the UI with the extra features for the administrator. These extra features are related with the two processes following this one. The latter is set to "owner", so the UI displays to the administrator that it has been connected as the owner. Both states are implemented in the LogInContext.

The process flow diagram is described below, with all the details and corresponding computations.

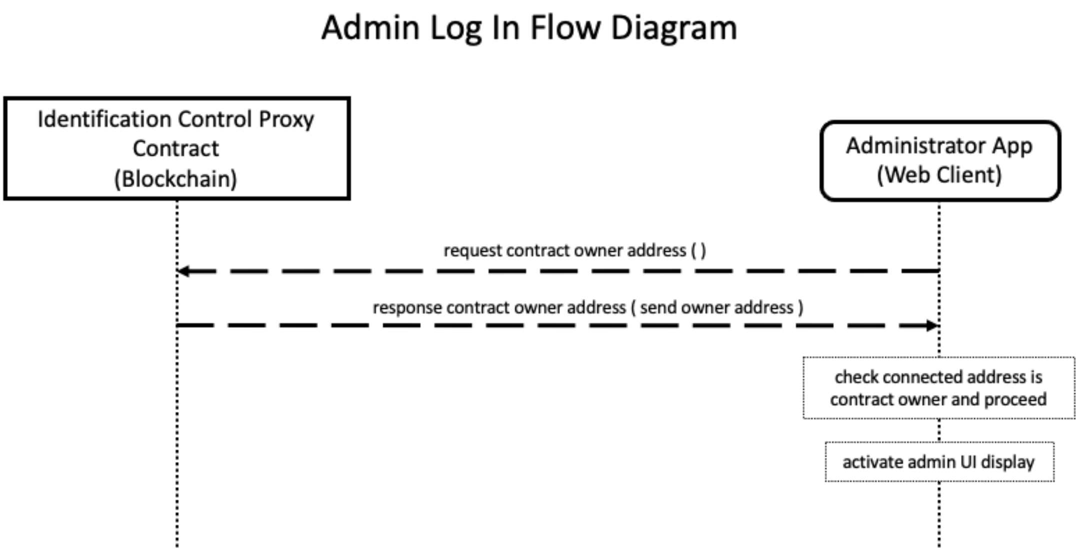


Figure 3.8. Admin Log In Flow Diagram

3.4.7. Admin Add User to Census Process

The admin add user to census is not a strictly necessary process for the system to operate if the census file is not empty.

It implements the functionality to add a user to the census file stored in the IPFS network. The UI view related to this process is implemented in the Admin component. It has three text inputs, one for the address to be added in the census, the other one for the ID related to the address and the final one, also a text input, for the Infura IPFS project secret, as this operation requires authentication in the Infura IPFS API. Finally, there is an interactive button to initialise the process.

About the UI states, the process does not change any major UI view state. It only updates the state parameters related to displaying in the administrator screen the Merkle tree root hash, the IPFS census file hash and the complete census. These parameters implemented as states can be found in the AdminContext.

The process flow diagram is described below, with all the details and corresponding computations.

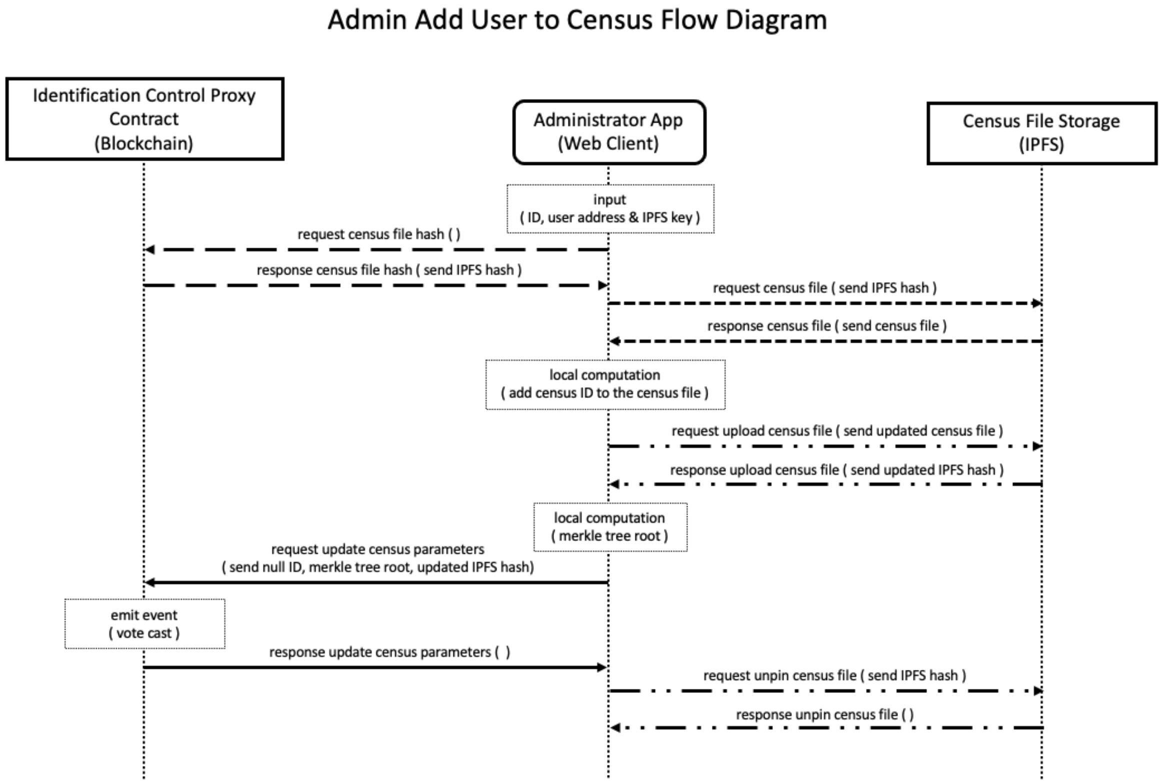


Figure 3.9. Admin Add User to Census Flow Diagram

3.4.8. Admin Remove User from Census Process

The admin remove user from census is not a strictly necessary process for the system to operate.

It implements the functionality to remove a user from the census file stored in the IPFS network. The component implementation is the same as the one in the add user to census process.

About the UI states, this process is as the previous one (Add from Census), it only updates view parameters related with the census administration. These parameters implemented as states can be found in the AdminContext.

The process flow diagram is described below, with all the details and corresponding computations.

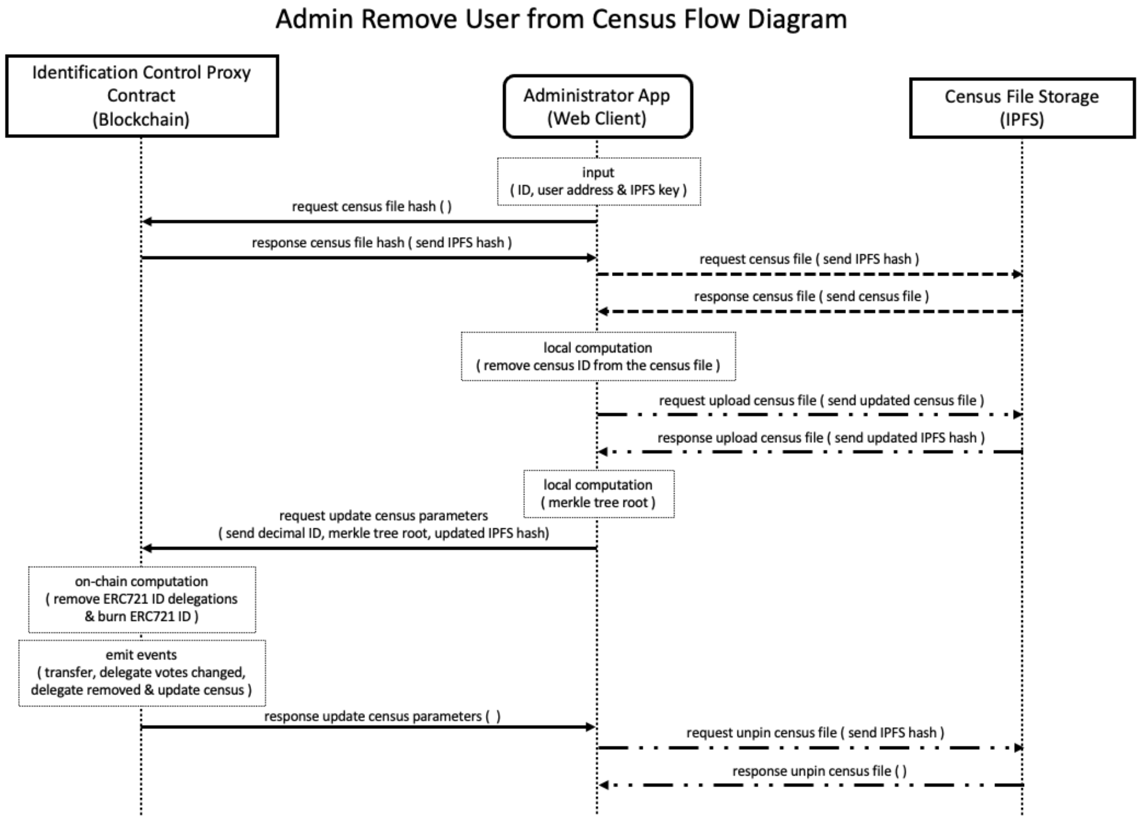


Figure 3.10. Admin Remove User from Census Flow Diagram

3.4.9. New Proposal Submission Process

This process is necessary to be executed at least once by some user to operate with the governance system, otherwise no proposal will ever be available in an active state.

It implements the functionality to submit a new proposal to the governance system in the Ethereum network. The UI view related to this process is implemented in the Governance component. It has one text input, to submit the description of the proposal, and a interactive button to initialise the process.

During this process no major UI state is modified. Only change in the screen view will be when the events emitted by the contract are updated, the new proposal will be displayed in the proposals list. The mentioned parameters list state is implemented in the GovernanceContext. As in the voteDelagete component, a view/hide functionality has been implemented here with the same design. In this case the state control variable is viewPC.

The process flow diagram is described below, with all the details and corresponding computations.

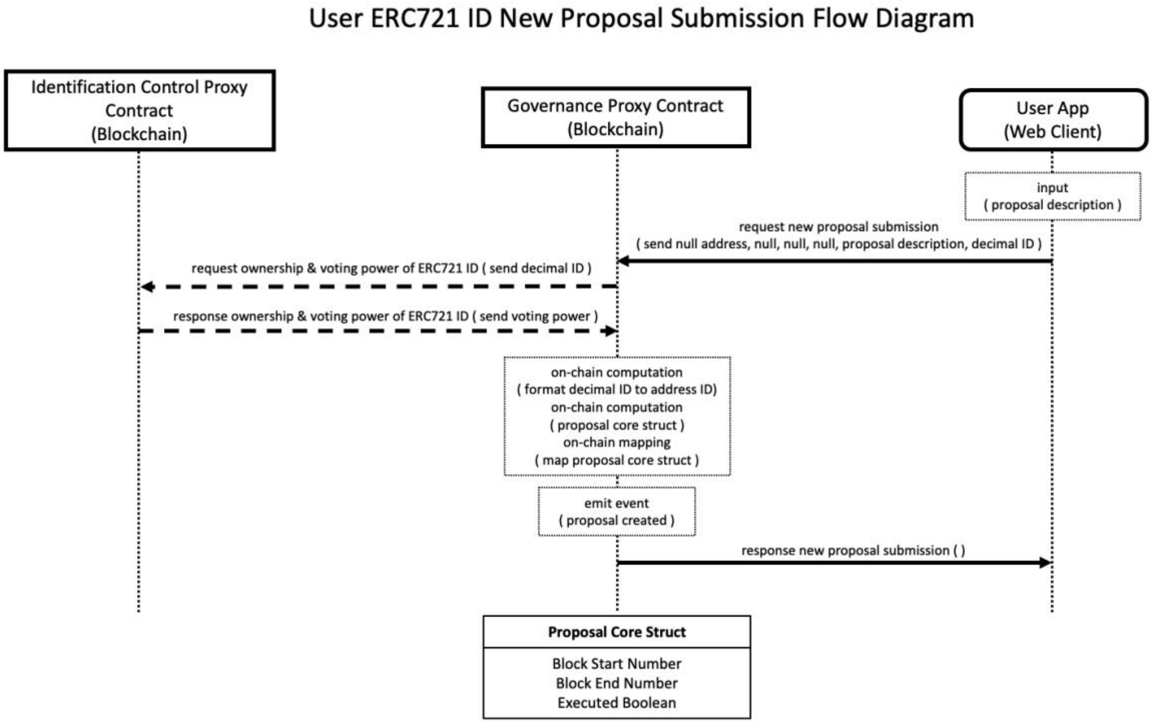


Figure 3.11. New Proposal Submission Flow Diagram

3.4.10. Vote Submission Process

This process requires for a proposal to be active but is it not strictly necessary for the system to operate, as even though if the user does not vote, the proposal will go though its designed stages in the governance contract.

It implements the functionality to submit a vote to an active proposal. The UI view related to this process is implemented in the Governance component. It has one React select component, that offers three options (for, against, abstain), to submit the user support for the proposal. Finally, it has an interactive button, to submit the vote to the Ethereum network. These interactive features are only viewable if the proposal is found in an active state.

About the UI states, no important UI view control state is updated. Only modification will be the proposals parameter hasVoted, which displays if either the user has voted or not in the displayed proposal. The parameter is updated whenever the proposals list parameter is updated again, through the request of the contract emitted events to the Infura API. Updated done at finalizing the vote submission when the page is reloaded. The mentioned parameters list state is implemented in the GovernanceContext.

The proposals are displayed from an object constructed from the emitted events by the governance control. The object contains the following parameters:

1. State of the proposal
2. Description
3. Creation, snapshot, end, and current blocks
4. Estimated remaining time
5. Total, for, against and abstain votes
6. Participation percent
7. Voting power / ID vote has delegated to

Note that the snapshot block is the block number on which the proposal vote was set to start.

The process flow diagram is described below, with all the details and corresponding computations.

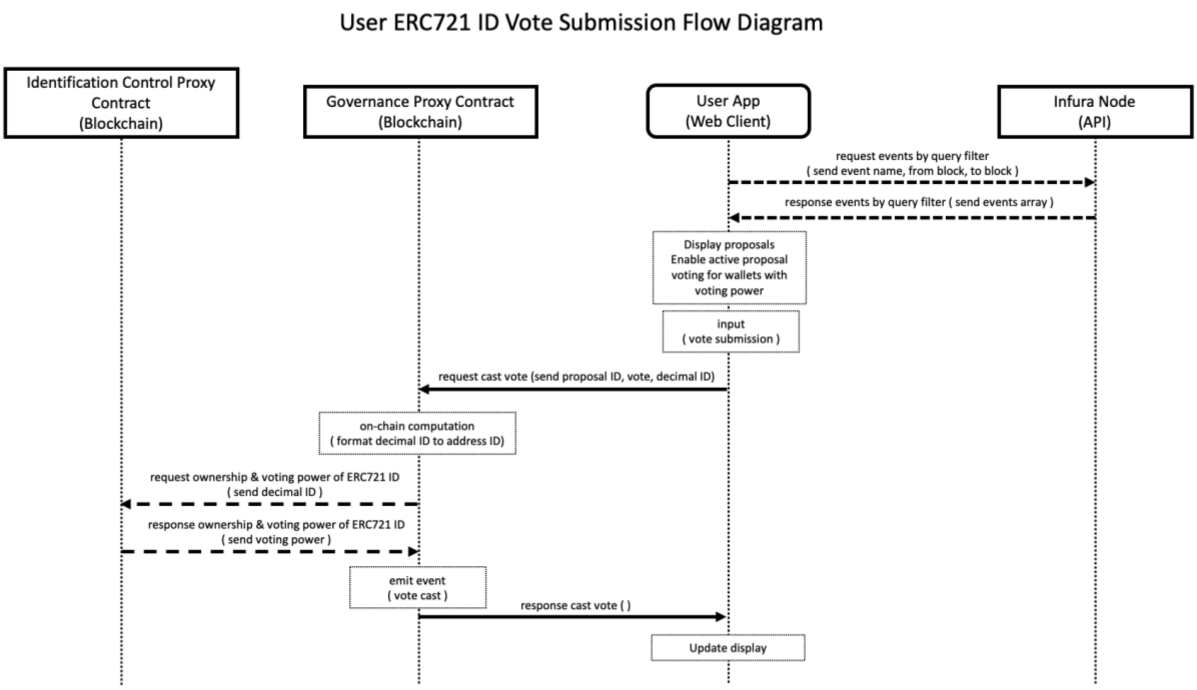


Figure 3.12. Vote Submission Flow Diagram

4. Project Results

In this chapter some data analysis from the project will be done. Given the nature of the project, being more focused towards a product development project rather than a purely research project, this section won't be very extensive. Some analysis on the designed IPFS storage systems will be made.

4.1. File Storage Usage Results

Firstly, a storage size calculation in bytes will be done for every census ID stored in the IPFS census file.

The encoding used is UTF-8, both in the web client and in the IPFS system.

Since the census ID is composed by and Ethereum standard address (64-character string) concatenated with a decimal ID (10-character string), the result for a census ID is a 74-character string.

As specified by the UTF-8 encoding standard, this results to the census ID being a word of 74 bytes size.

As the IPFS file is treated in a CSV format, every census ID will be followed by a comma, making it a 75 bytes size word for census ID. Even though the last census ID of the file won't contain that comma, the approximation will be done using a 75 bytes word per census ID.

The results are as follows:

Number of census entries [units]	Census file size [kbytes]
1	0.075
5	0.375
10	0.75
50	3.75
100	7.5
500	37.5
1000	75
5000	375
10000	750
50000	3750
100000	7500
500000	37500
1000000	75000

Figure 4.1. IPFS Storage Analysis Results

Given this design and that every IPFS object can sustain up to 256kbytes of information, one IPFS object would be able to store up to 3413 census IDs entries.

5. Budget

The materials and resources used in the development of this project are included in budget.

- Web hosting: Hostinger is the web hosting provider selected for the website's allocation, with a monthly plan of 1.99€. After a research phase, it was determined that this is the most realistic service in terms of price.

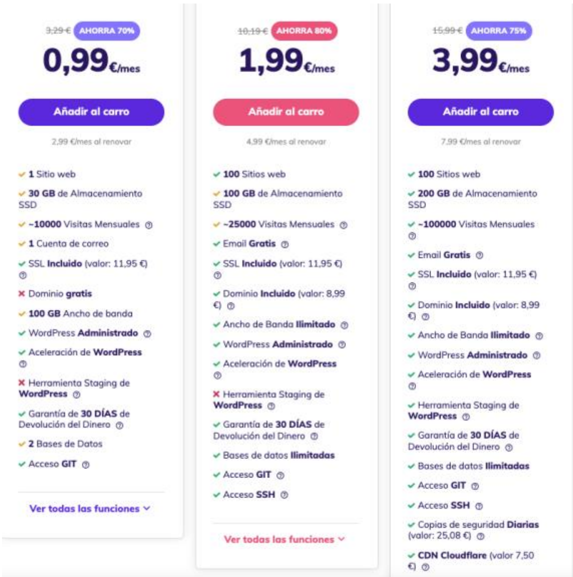


Figure 5.1.Web hosting pricing

- Infura: Because it is one of the more secure companies running EVM nodes, Infura was picked as the Ethereum network's connection. For the time being, the free option will satisfy the system requirements.

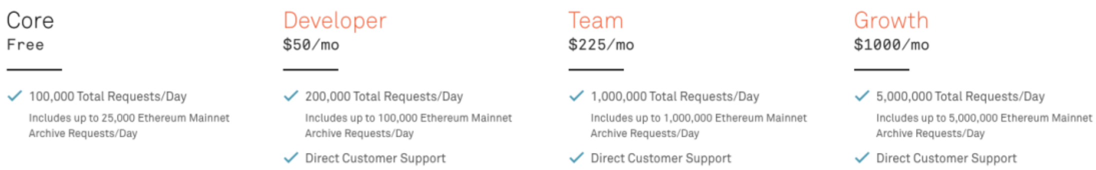


Figure 5.2. Infura Ethereum services pricing

- IPFS: The free plan would be the one the suits better the needs for the project by now.

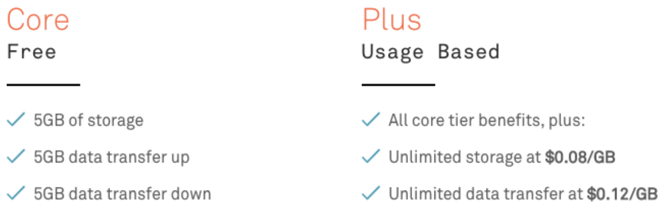


Figure 5.3. Infura IPFS services pricing

- Ethereum network: The ether and gas price on May 28, 2022, which is respectively 1644€ and a gas price of 25.66 gwei, will be used to generate an estimation of the gas fees to be paid.

The table below illustrates the average amount of gas wasted throughout the deployment of the two system contracts

Contract	Method	Min	Max	Avg
Methods				
GovernanceUpgradeable	castVote	-	-	69464
GovernanceUpgradeable	initialize	-	-	186347
GovernanceUpgradeable	propose	-	-	90775
IdentificationRegistryContract	delegate	-	-	128218
IdentificationRegistryContract	initialize	-	-	252304
IdentificationRegistryContract	whiteListMint	292561	303727	298144
Deployments				
GovernanceUpgradeable		-	-	1783955
IdentificationRegistryContract		-	-	3399756

Figure 5.4. Smart contract gas fees

In total, 5.622.442 gas was required for both deployments. To compute the number of gas required for the deployment it is needed to sum up the average gas from deployment and the initialize methods. From the following formula the total price in euros can be computed for the deployment of the contracts:

$$\text{Gas €} = \text{Gas Cost} \div 1 \text{ billion} \times \text{Gats Price in Gwei} \times \text{ETH Price €} = 239.12\text{€}$$

The cost of gas is also described for a propose, castVote, delegate and registration transactions. From the defined formula defined above, the cost of each transaction is:

- Propose: 4.17 €
- CastVote: 2.93 €
- Delegate: 5.4 €
- Registration: 12.58 €

Ideally the project would be better deployed on a lower gas fees network. Usually, Proof of Stake blockchains do have this characteristic, so with Ethereum migrating to Proof of Stake, the budget will be considerably improved. Another option that also use the EVM and would make it easy to migrate to is Polygon, an Ethereum layer 2.

Salaries: For a total of 300 hours, two junior engineers worked on this project. With a 12-euro-per-hour wage, the total comes to 7200€.

6. Conclusions & Future Development

To conclude the thesis, some conclusions and future development statements will be discussed in this section.

When it comes to the conclusions, they could be divided into two distinct parts, the technological and the human parts.

Probably it could be said that the main takeaway from the development of the project has been the human part, more precisely, the behaviour we have as a society and this behaviour is altered when power and money are at stake. It was not expected that this would be as challenging as the technological development part of the project.

Developing a governance protocol is very delicate work. It is a system that will have to deal with the best and worst of human nature and the collective behaviour we have as a species. It is extremely hard to make a system that equally voices out everyone's opinion since there are no two equal persons. At the same time, it is even harder to give a greater voice to those that are extremely qualified and will provide the greater common good while ensuring that the system will not fall into a plutocracy or a polyocracy.

Although it has not been mentioned, because this is a final degree engineering thesis, during this project lots of philosophical thinking has been needed on to what would be the best approach to implement a system that stands within the earlier statements. During the process, even greater dilemmas appeared. For example, when the majority stands together but is wrong towards deciding for the common good, would that be wrong? History has shown us over and over that sometimes, great advancements for humanity were abolished simply because the majority did not understand and consequently feared an idea.

After giving it a lot of thought, the conclusion so far has been that the current voting systems that democratic countries rely on are the best approach for now. These voting systems, need to incentivise the individuals to learn and educate themselves about the decision-making processes and proposals. It must be a system that makes people feel part of it since this will make them be willing to proactively take part. Which is a scenario that has already been seen in some decentralized communities relying on blockchain networks.

Therefore, we have chosen the aforementioned approach in the project, as we strongly believe that this new governance system will provide the opportunity to make people feel involved in the decision-making process. The system will make it possible to organise these rituals more often thanks to the characteristics of the new hosting infrastructure besides giving out more freedom to create and organize into new structures of power and decision-making delegations.

Regarding the technical part, it has been really challenging the development of the user interface, as it has been the first time I've done so. Nevertheless, discovering React and Tailwind has been a fun journey.

One of the main takeaways from the project has been the unexpected complexity it involves implementing a web user interface. Although React helps a lot to keep the logic implementation of functions separated from the HTML and CSS, to orchestrate everything and make it work requires a well-planned design.

Another thing taken away from the technical part is the fact that a blockchain network opens a new paradigm for the typical client-server application. As it has been seen in the develop project, it allows to develop a client-server app without the implementation of the classic server.

This new type of applications offers more robustness and formalism in terms of security. There is no need to implement so many security measures while keeping a high guarantee. But as everything, these benefits come at a cost, which is the high operating costs and latency between client and the network. So, it could be concluded that only applications that really require high security standards would be worth developing using this type of architecture.

I believe think that the architecture design if the app has been well chosen given the security requirements a voting system has. But it also must be said that lots of optimization must still be done, as many of the checks must be done twice to avoid having to pay fees for transactions that will fail. Also, the operating costs are high for simple transactions, as seen in the budget section, which makes it hard for the project to be adopted for commercial use.

Concerning future developments, this project is set to be just the start of a long research journey. This research thesis has served as a good foundation layer to start building on top and at the same time, it has proven that the system could work. The system is scalable and secure.

The following steps intended to take are improvements on various of the implemented systems.

When it comes to the registration and identification system, the inclusion of an identity verification tool as the Spanish National Identity system for electronic signatures would be a significant improvement toward the general adoption of electronic voting systems. This would allow for the census and identification process to take place under a system already used on a national scale for electronic identity verification by providing better security and trust guarantees. The combination of this type of tools with a public blockchain electronic election system would allow to truly replicate the current election voting process of an electronic nature, reducing costs and improving accountability, security and reliability.

In relation to the registration system, it will be the inclusion of multi-signature wallets for administrative purposes that will add an extra layer of security to these wallets that hold such heavy responsibility. The security of the multi-signature wallet could lead to the development of community funds treasuries and consequently, community users could take part in decision-making processes to administrate these funds.

When it comes to the governance system, future developments have also been identified. For the voting process, the inclusion of a privacy feature using zero-knowledge proofs technology and a coercion-resistant system, inspired by a research paper titled “Coercion resistant electronic-elections", where a cryptographic system on which users can override their earlier vote is defined.

Next is the implementation of roles within the governance system, making use of the ERC721 operability and functionalities. As a result, distinct roles can be assigned to the community members or even create community groups within a community.

Another future implementation will be to make use of the Ethereum Gas Station, for enabling the protocol to pay for the users' transactions gas fees. This would allow increasing the engagement of the users with the voting process. Making users pay gas fees for voting would dissuade a substantial number of users from participating.

Finally, to conclude, some extensive optimization work must be done to make the project more affordable and attractive for widespread use. This is one of the main factors that would make people not adopt these systems. Optimization will be implemented in data storage systems (data compression), in system internal variables (optimizing IDs to uniquely identify while using the smaller number of bytes possible), and in blockchain operations, both communication and computation, to reduce gas fees and resources consumption.

Bibliography

[1] Haber, S., Stornetta, W.S. “How to time-stamp a digital document”. J. Cryptology 3, 99–111 (1991). DOI: https://doi.org/10.1007/BF00196791

[2] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". *Bitcoin Whitepaper*. [Online] Available: https://bitcoin.org/bitcoin.pdf. [Accessed: 15 March 2022].

[3] Markus Jakobsson; Ari Juels. "Proofs of Work and Bread Pudding Protocols". Secure Information Networks: Communications and Multimedia Security. Available: http://www.arijuels.com/wp-content/uploads/2013/09/JJ99.pdf. [Accessed: 12 June 2022].

[4] Aggelos Kiayias, Alexander Russell, Bernardo David, Roman Oliynykov. "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol". *Proof of Stake White Paper*. [Online] Available: https://app.box.com/s/eui5ayv98rw5m9ysvtnkrirlm5wpm7og. [Accessed: 3 April 2022].

[5] Dr. Gavin Wood. "Ethereum: A Secure Decentralised Generalised Transaction Ledger". *Founder, Ethereum & Parity*. [Online] Available: https://ethereum.github.io/yellowpaper/paper.pdf. [Accessed: 29 April 2022].

[6] Sabih Rehman. "Ethereum: A Next Generation Smart Contract and Decentralized Application Platform". *Medium*. [Online] Available: https://medium.com/@sabih811/when-and-where-blockchain-comes-9a2ffe59edea/. [Accessed: 4 April 2022].

[7] Vitalik Buterin. "Ethereum: A Next Generation Smart Contract and Decentralized Application Platform". *Founder, Ethereum & Parity*. [Online] Available: https://ethereum.org/en/whitepaper/. [Accessed: 3 April 2022].

[8] Ethereum Foundation. "Ethereum Developer Resources". [Online] Available: https://ethereum.org/en/developers/. [Accessed: 30 May 2022].

[9] Horizen Academy. "UTXO vs Account Model". [Online] Available: https://academy.horizen.io/technology/expert/utxo-vs-account-model/. [Accessed: 3 May 2022].

[10] Vitalik Buterin. “Proof of Stake FAQ”. [Online] Available: https://vitalik.ca/general/2017/12/31/pos_faq.html. [Accessed 13 April 2022]

[11] InterPlanetary File System. “IPFS Official Documentation”. [Online] Available: https://docs.ipfs.io/. [Accessed 24 April 2022]

[12] Juan Benet. “IPFS – Content Addressed, Versioned, P2P File System (DRAFT 3)”. [Online] Available: https://ipfs.io/ipfs/QmV9tSDx9UiPeWExXEeH6aoDvmihvx6jD5eLb4jbTaKGps. [Accessed 25 April 2022]

[13] Cloudfare. “Serving Cloudflare Pages sites to the IPFS network”. [Online] Available: https://blog.cloudflare.com/cloudflare-pages-on-ipfs/. [Accessed 15 June 2022]

[14] Hash Function Wikipedia. “Hash Functions”. [Online] Available: https//en.wikipedia.org/wiki/Hash_function. [Accessed 5 May 2022]

[15] Shorouq Alansari. "A Blockchain-based Approach for Secure, Transparent and Accountable Personal Data Sharing". M.S. thesis, Faculty of Engineering, Science and Mathematics School of Electronics and Computer Science, University of Southampton, Southampton, England, 2020.

[16] IDEN3. “Merkle Tree: A visual introduction”. [Online] Available: https://blog.iden3.io/merkle-trees-visual-introduction.html. [Accessed 5 June 2022]

[17] Santiago Palladino, Francisco Giordano “EIP-1967: Standard Proxy Storage Slots”. Ethereum Foundation. [Online] Available: https://eips.ethereum.org/EIPS/eip-1967. [Accessed 17 June 2022]

[18] Open Zeppelin “Proxy Upgrade Pattern”. [Online] Available: https://docs.openzeppelin.com/upgrades-plugins/1.x/proxies#the-constructor-caveat. [Accessed 6 May 2022]

[19] Vocdoni. “Vocdoni docs”. [Online] Available: https://docs.vocdoni.io. [Accessed 27 April 2022]

[20] Wikipedia UTF-8. “UTF-8”. [Online] Available: https://en.wikipedia.org/wiki/UTF-8. [Accessed 20 June 2022]

[21] Vitalik Buterin “Blockchain voting is overrated among uninformed people but underrated among informed people”. [Online] Available: https://vitalik.ca/general/2021/05/25/voting2.html. [Accessed 7 June 2022]

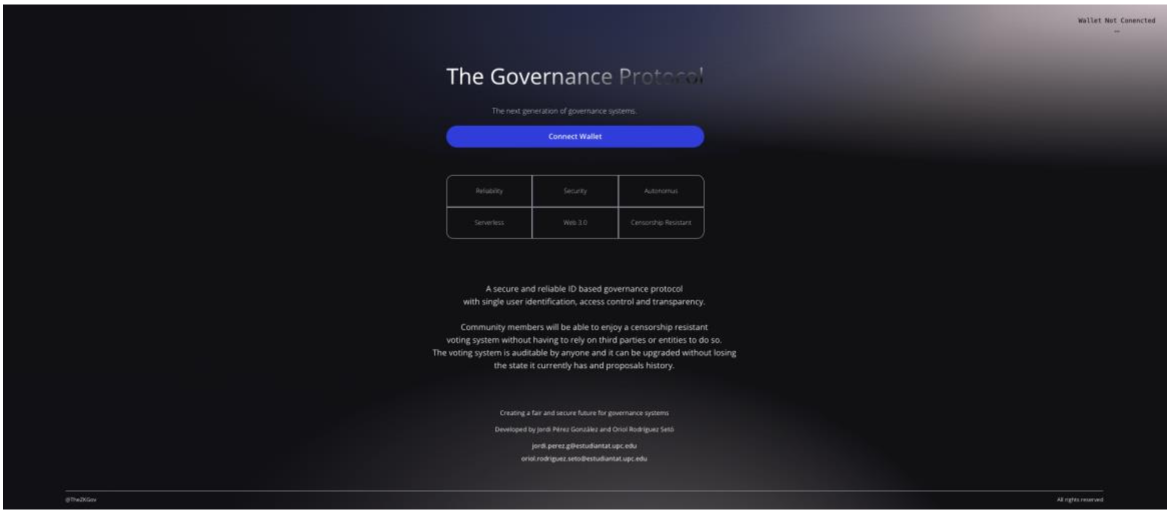
Appendices

All the project web client code can be found in the following GitHub repository

Project GitHub: https://github.com/uri011/TheGovProtocol

Front End Screens

Landing Screen

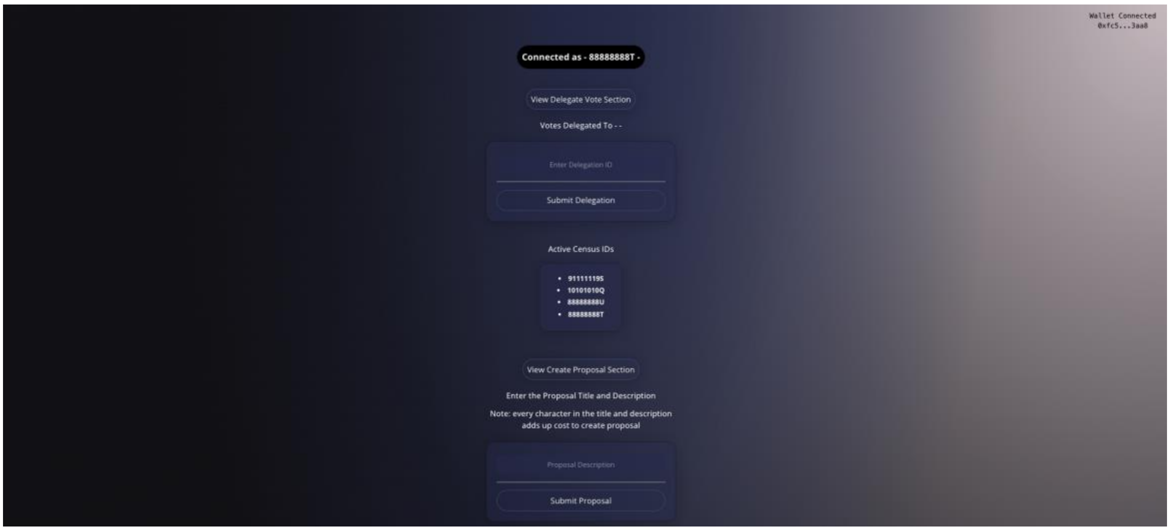


Composed by the Navbar, LogIn and Footer components.

Log In & Transfers Screen

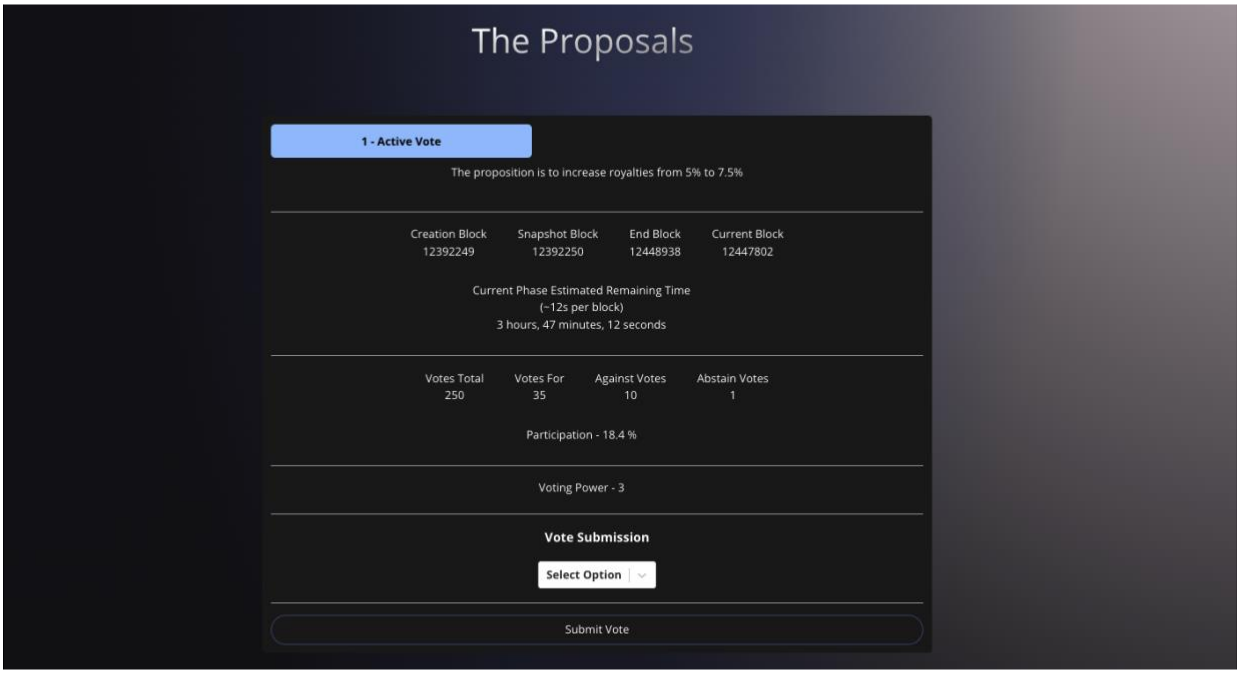
Composed by the Navbar and LogIn components.

Vote Delegation & Proposals Submission Screen



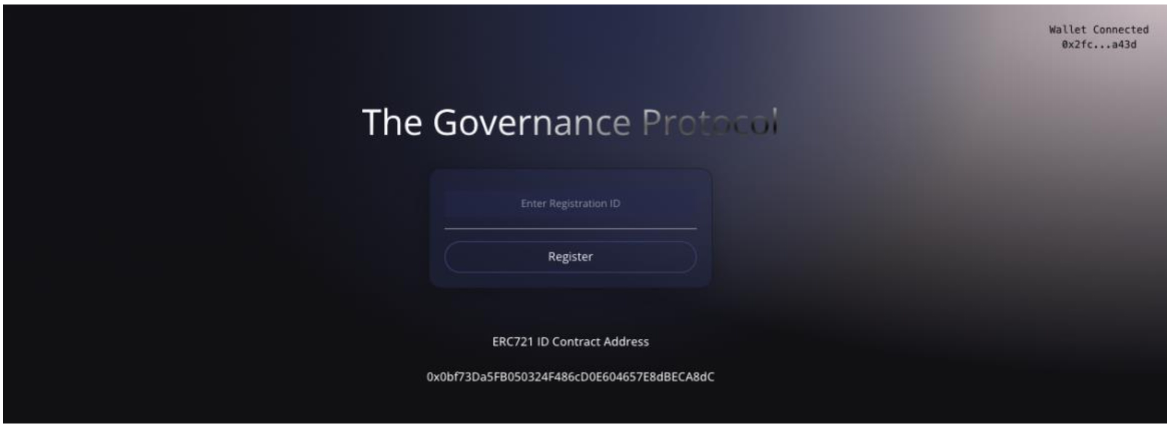
Composed by the Navbar, ProposalsCreate and VoteDelegate components.

Vote Proposals Screen



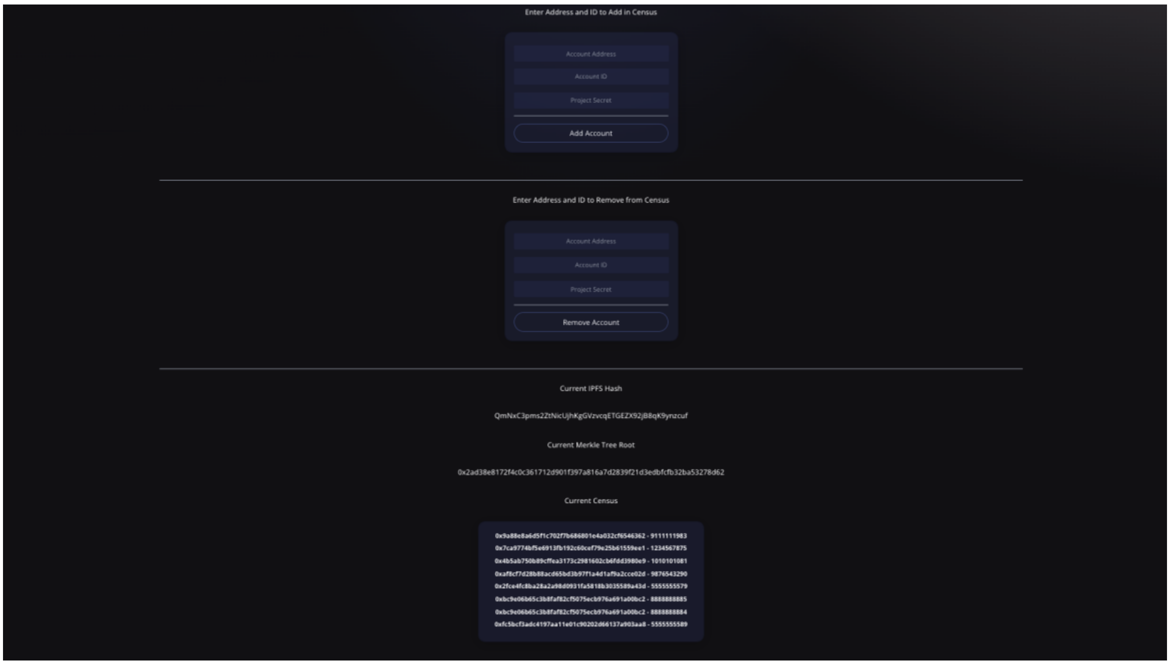
Composed by the Proposals component.

Registration Screen



Composed by the Navbar and LogIn components.

Administrator Screen



Composed by the Admin component.

Glossary

A list of all acronyms and the meaning they stand for.

- **ID:** Identification
- **PoS:** Proof of Stake
- **PoW:** Proof of Work
- **Dapp:** Decentralized application
- **EIP:** Ethereum Improvement Proposal
- **UTXO:** Unspent Transaction Output
- **DDoS:** Distributed Denial of Service
- **Wei:** Smallest possible division of an Ethereum token
- **Gwei:** Wei by a 10^9 factor
- **EVM:** Ethereum Virual Machine
- **IPFS:** InterPlanetary File System
- **URL:** Uniform Resource Locator
- **Merkle DAGs:** Merkle Directed Acyclic Graphs
- **DHTs:** Distributed Hash Tables
- **ERC721:** Non-Fungible Token Standard
- **UI:** User Interface
- **ABI:** Contract Application Binary Interface
- **IDE:** Integrated Development Enviroment
- **CSV:** Comma Separated Values
- **UTF-8:** Unicode Transformation Format – 8 bit