INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




# Governança Extensivel

**HENRIQUE FILIPE CHUVA**
Outubro de 2022

POLITÉCNICO
DO PORTO


# Extensible Governance

Henrique Filipe Chuva

**A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Computer Systems**

**Supervisor: Nuno Bettencourt, PhD.**
**Co-Supervisor: Gonçalo Prendi**

Porto, October 15, 2022

# Dedicatory

To my parents, who never doubted I would be able to cross the finish line, for always supporting me unconditionally. To my everything for believing in me, and giving me strength and motivation to keep pushing forward while putting up with me and my non-sense. To all my closest friends for not only supporting me through this journey, but for all the help they gave every time it was necessary.

# Abstract

With a growing number of institutions investing in crypto assets every year, along with new blockchains and protocols being developed, services meant to manage and safely store such assets called custody services are becoming more and more sought after, and organizations are looking to improve their custody services by developing other services along side it.

One service becoming more and more important for institutions is governance, allowing to visualize and vote for proposals that can significantly impact the assets these same institutions hold and care about. One of the main issues with this growing interest in governance is the time taken to support new protocols and blockchains, as many of them differ in the way they work. For this reason, the following project has the objective of improving the process to support new protocols and blockchains by developing an abstracted service that can be deployed on top of any blockchain or protocol with minimal configuration and code modification, while supplying a standardized Application Programming Interface (API). This project was developed along side Anchorage, a company that provides custody services since 2017. This paper describes in greater detail the process and development of the project, including methodologies used to better analyse the current blockchain state of the art at a technical level, a design solution that fits our objectives and reaches our goals, the implementation process, experimentation methodology and our conclusions.

We developed a code based solution aimed at implementing the various required functionalities, further developing a proposal for an Ethereum EIP which aims at standardizing the current governance environment. Although issues were found when analyzing possible baseline metrics, we planned out an experimentation methodology which allowed us to test our solution by implementing different projects and analyzing the results. We finished the project with success achieved, with our service solution achieving both good results and better than the baseline, while also having an EIP proposal in which we believe can continue development with the Ethereum community itself.

**Keywords:** Blockchain, Ethereum, Governance, Solidity, Golang, Smart Contracts, Defi

# Resumo

Com um número crescente de instituições a investir em bens e moedas digitais, e novas *blockchains* e protocolos a serem desenvolvidos regularmente, serviços de gerência e segurança destes bens (serviços de custódia) estão se a tornar cada vez mais procurados, e organizações que fornecem estes mesmos serviços encontram-se a tentar desenvolver novas funcionalidades e até mesmo novos serviços que complementam a custódia de bens.

Um destes novos serviços é chamado de governança, e permite a instituições procurar, verificar e votar em propostas que podem ter grandes impactos nos bens que estas instituições possuem. No entanto, com novos protocolos e blockchains a serem desenvolvidos a um ritmo crescente, muitas das organizações que pretendem fornecer serviços de governança encontram-se com dificuldades em suportar estes novos protocolos de uma maneira rápida e de baixo custo, visto existirem sempre diferenças nas suas funcionalidades. Assim, o objetivo deste projeto é melhorar o processo de suportar novos protocolos ao desenvolver um serviço abstraído que possa ser implementado mais rapidamente e facilmente, diminuindo assim os custos do mesmo, facilitando a configuração e fornecendo uma API estandardizada.

O projeto foi desenvolvido em conjunto com a Anchorage, uma organização que fornece serviços de custódia desde 2017, e com este projeto iremos descrever pormenorizadamente o processo de desenvolvimento do serviço mencionado, fazendo uso de diferentes metodologias que nos permitem analisar o estado da arte atual da tecnologia *blockchain*, e também projetar e definir possíveis soluções que permitam atingir os objetivos definidos, explicar o processo de implementação de possiveis soluções, metodologias de experimentação e conclusões.

Durante o projeto desenvolvemos um serviço no formato de software com o objetivo de implementar as várias funcionalidades definidas como objetivos, criando uma possível proposta para um EIP de Ethereum com o intuito de estandardizar o ambiente de governância atual. Uma metodologia de experimentação simples foi também desenvolvida com o intuito de testar a nossa solução ao implementar vários projetos e analisando os resultados. O projeto foi finalizado com sucesso, obtendo resultados satisfatórios. Acreditamos também que a proposta EIP desenvolvida é promissora e será melhorada quando partilhada com a comunidade de Ethereum.

# Acknowledgement

I would like to thank Anchorage for not only the amazing internship I was given, but also all the help provided through out the six long months by everyone involved, specially my coordinators Gonçalo Prendi and Daniel Gonçalves, who not only helped my when necessary, but also strived to better understand the entire project, becoming a crucial part of this project. A lot was learned, but there is still a lot more to learn.

# Contents

# List of Figures

# List of Tables

# List of Source Code

# List of Acronyms

| | |
|---|---|
| ABI | Application Binary Interface. |
| AHP | Analytic Hierarchy Process. |
| API | Application Programming Interface. |
| DAO | Decentralised Autonomous Organisations. |
| Dapps | Decentralized applications. |
| DeFi | Decentralized Finance. |
| DSR | Design Science Research. |
| ECDSA | Elliptic Curve Digital Signature Algorithm. |
| EIP | Ethereum Improvement Proposals. |
| EVM | Ethereum Virtual Machine. |
| GETH | Go Ethereum. |
| HSM | Hardware Security Module. |
| NFT | Non-fungible token. |
| QFD | Quality Function Deployment. |
| RPC | Remote procedure call. |
| UTXO | Unspent Transaction Output. |
| XML | Extensible Markup Language. |
| YAML | YAML Ain't Markup Language. |

# Chapter 1

# Introduction

Since the launch of Bitcoin in 2009, cryptocurrencies have become more valuable each year, with trading sites stock trading being mimicked by sites, creating an all-time high value industry, highly dependent on the volatility of said cryptocurrencies in order to make money. People started buying these currencies, also called assets, when their price are low, and then selling them back to the market when their price rises, making a profit based on how much the price as changed. Although these prices raises and drops are mostly started by news related with the crypto world (such as changes to a specific currency for example), it quickly became apparent that the general public reaction had a much greater impact on an asset value. With an ever growing number of new blockchains and cryptocurrencies, many of which organizations all over the world own and invest in, it has become significantly harder for these same organizations to keep track of all the different assets owned across the different protocols, more specifically quantity owned, their value, and the governance situation of the same protocols on which they invest in.

Since all of these factors are of great importance for each organization, new platforms with the purpose of centralizing various blockchains and protocols in one place are emerging, with services that range from governance to staking, custody, trading and much more. These same platforms have had a tendency to become cryptocurrency banks, lending assets to other organizations, which in turn generates interest for the original owner of these assets, if he so wishes or allows. This work is developed as a master dissertation for the Masters in Computer Engineering (of the Computer Engineering Department), Information and Knowledge Systems branch, of the School of Engineering (ISEP) of the Polytechnic of Porto (IPP), alongside a professional internship taking place at Anchorage[1], a company that started in 2017, specializing in crypto assets custody services, and aims at further studying and analysing the current governance environment in blockchains. The project was mainly focused on the Ethereum blockchain, as an in depth study of a single technology allows us to develop a more well-thought solution. That said, this paper does provide two different solutions that can be easily applied to other governance supporting blockchains.

Anchorage is a digital asset platform directed at institutions, and provides a multitude of services that lets investors hold and use cryptocurrencies safely and easily. Their services spread across multiple areas, such as custody, trading, staking, governance and financing services. To better explain the scope of this thesis, we will focus on one service, which is the governance service. Anchorages' governance services allows institutions to participate on issue or development related decisions that affect the protocols those same institutions care about. This is done via on-chain voting on proposals which

might be of interest. Most of the accepted protocols are Decentralized Finance (DeFi) oriented protocols, which may focus either on general cryptocurrencies or stablecoins.

## 1.1 Problem

Currently Anchorages' governance service supports many different protocols, such as the Maker protocol[1], Compound[2] and Aave[3], which all run on the Ethereum[4] blockchain, but also support other blockchain technologies, such as Celo[5]. Even though the list is constantly expanding based on client demand, at the moment the service in charge of implementing said protocols is not fully abstracted due to lack of similarities between governance projects, with each new accepted protocol being integrated separately, and demanding custom code implementations for each main requirement, such as acquiring and voting on already existing proposals.
In turn, the development and integration time for these protocols becomes rather long, with a much more complex workflow and no kind of standardization. Furthermore, this also leads to very different implementations for each protocol, which leads to much harder maintenance and possible further development of new customer driven features. Thus, the problem we are trying to solve is the fact that the process of supporting new protocols is not streamlined, requiring a long development time from the beginning, with no template nor abstraction available.

## 1.2 Goals

The main objective of this project is to study and analyse the current ecosystem of Ethereum governance projects and their abstraction limitations, in order to develop a service, which allows for easy support and deployment of new Ethereum governance protocols/projects, with minimal changes necessary, allowing Anchorage to streamline the implementation of new projects into their governance service. Although we provide two possible solutions, they both aim to:

**G1)** Abstract the voting process of governance projects;

**G2)** Abstract the delegation process of governance projects;

**G3)** Reduce the cost of supporting new governance projects;

## 1.3 Research Questions

Now that we have defined our problem, and before we can continue with our research, we must define our research questions. These questions will better define the scope of our research, and also help us focus on specific details when needed. First and foremost, the main research question we have identified is:

**RQ1)** Can we abstract the main governance functionalities, such as voting and delegating?

[1] https://makerdao.com/en/
[2] https://compound.finance/
[3] https://aave.com/
[4] https://Ethereum.org/en/
[5] https://celo.org/

**RQ2)** Can we develop a Framework to streamline the support of new governance projects that can be deployed on Ethereum?

**RQ3)** What limitations must we work with in order for our Framework to work, if any?

**RQ4)** How can we evaluate our Framework when adding support for new protocols in terms of development cost and time?

We can further separate these four research questions in three separate groups. The first group, containing the question RQ1, pertains to the current Ethereum governance ecosystem, mainly on current common patterns and design choices. RQ2 and RQ3 pertains heavily to the architecture, layout and development of the Framework itself, while the final group, containing the question RQ4, pertains to the experimentation and evaluation of the resulting Framework.

## 1.4 Research Methodology

The chosen research methodology for this project is called Design Science Research (DSR) methodology. As the name implies, its based on a design science paradigm, and it will be further explained and analysed on chapter 2. This methodology provides a general process iteration which we will follow, including the identification of the problem and our motivation to fix the problem, designing a solution that accomplishes the defined goals, and after development the demonstration and evaluation of the final product. Further more, DSR also provides a set of guidelines which allows us to plan and evaluate the previously mentioned steps.

## 1.5 Hypothesis

Our main hypothesis of concern is how can we develop a service that abstracts the communication and necessary interactions with Ethereum governance projects. The final result of this hypothesis creates another hypothesis in itself, being if said framework is developed, how much will it decrease the time and costs of development for new Ethereum governance protocol acceptance.

## 1.6 Thesis Structure

This paper is structured in eight different chapters that follow the previously mentioned research guidelines. The current chapter is the introduction, where we presented the problem to solve, the context and importance of this problem, and the hypothesis we want to prove with a set of defined goals to achieve and research questions to answer. Chapter two consists of an in depth research on the current state of the art, focusing on already existing related literature, the state of the underlying technology and recent developments in the area. Following is chapter three, which focus on the business value of the project, both through the business perspective and end user perspective, and also on the requirements of the project. Chapter four focuses on designing different possible architectures for the project in hand, starting with a prototype design, possible alternatives for the implementation of this design, and the minimal requirements. Chapter five describes the implementation of the prior designed prototype, the technological stack, and a detailed explanation on how the system was developed. Chapter six further

presents and explains a possible second solution in the format of a standard proposal, which aims at correcting many of the problems faced during chapter four and five. In chapter seven we assess the final implementation, defining the experiment indicators and analyzing the experimentation results. Finally, in chapter eight we revisit the original research questions, analyzing the obtained answers, possible contributions and limitations, and comment on possible future work to further develop the technology.

# Chapter 2

# State Of The Art

In this chapter we are going to review already existing literature on the subject we are studying. We will first do a systematic review on our chosen research method, to contextualize the reason for why we chose it in the first place, the theory behind it, and a small comparison with another method. The following sections will focus on the blockchain itself, the history of the technology, and how governance fits in. Finally we will present and contextualize many of the technologies and tools we will be using for the development and implementation of the project.

## 2.1 Related Work

In this section we present existing literature on the blockchain technology, including how it works and some important technical details, analysing the reasons and necessities for its creation and development. We also analyse some blockchain examples that are considered of importance for the growth of the technology itself, and finally how governance fits in the scope of this technology and how it is defined. It is important to note that currently, most literature is focused on theoretical propositions, mainly related to possible uses of the technology in different areas and industries. Although this is of importance, and it allows us to understand how much this technology can and will grow, the lack of technical information is nonetheless concerning, as understanding the inner workings of bitcoin and ethereum, such as code base and code flow, is of the utmost importance for our project, as it allows us to understand how to go about our objectives, given that we need to work directly with these same inner workings. As such, when important literature is either missing or incomplete, we resort to both online web pages that we consider trustworthy. Any and all of the knowledge and information we pretend to use and share on this paper was mostly verified and tested against the original code base of any technology we talk about, such as bitcoin and ethereum. Our state of the art aims not only at contextualizing the current state of all the technologies we intend to use, but also setting a literature base line of knowledge when it comes to more technical details related to blockchain technologies.

### 2.1.1 Blockchain

The term "Blockchain" can be found being used loosely in plenty of modern literature, either referencing the technology in general, the category on where it falls in, or even specific implementations. That said, Beck et al.[3] characterizes it as a class or part of technologies, often called Distributed ledgers, further describing it as a decentralized, transactional database technology, that allows for validated, tamper-resistant transactions consistently across a network with a large amount of nodes. Yaga et al.[24]

additionally defines four key characteristics of the technology, such as: i) unlike traditional databases, transactions cannot be modified nor deleted, being append only to provide a full transnational history; ii) the blockchain is cryptographically secure, ensuring that the data is tamper proof and attestable; iii) the ledger provides full transparency by being shared amongst multiple network nodes; iv) the blockchain is fully distributed, allowing the scaling of the number of nodes on the network, and as such the scaling of the network itself, which in turn makes it more resilient to attacks, by reducing the impact bad actors can have on the consensus protocol.

A blockchain consists of a chain of data blocks (data packages). Each block is composed primarily of transactions, but also contains a timestamp, the hash value of its content, the hash of the previous block and a nonce. With enough transactions a new block is created, but it is only added to the existing chain after it is validated by the network using cryptographic means. After being validated and added, the chain is extended, thus representing a complete ledger of transactional history [19]. The hash values (results of an hash-function) mentioned previously are unique and assure that the blockchain is tamper-proof. An hash-function is an easily computable function from the set of all finite binary strings to a set of binary strings of some fixed length, being able to compress an input string into hashcode. It must also be assured that the hash function is one-way only, making it impossible to revert back to the original input, and that the chance of two inputs resulting in the same output (collisions) is close to none [9, 18]. The use of hashing ensures that changes to a block in the chain would immediately change the respective hash value, and as such make the following blocks invalid, as all of them would have the wrong hash values themselves.

### 2.1.2 Bitcoin

There are already a multitude of blockchain use cases reported and implemented, such as identity management, land registration, notarization, supply chain traceability, healthcare, education, and many others [22], but the blockchain considered to be the start of this trend was the Bitcoin blockchain and cryptocurrency, and to this day the biggest area of implementation is still digital currency and payments. Bitcoin was first introduced in 2008 in a paper by Nakamoto [17], presented as a peer-to-peer electronic cash system, which would allow for online payments, directly between two parties without the need for a financial institution as a proxy. Although digital signatures provided part of the solution for this, one of the main focus of the paper was to prevent double-spending by implementing a proof-of-work based consensus algorithm. The only caveat with Bitcoin is the fact that the blockchain itself is single purpose, meaning that the only purpose of this blockchain was to support the Bitcoin cryptocurrency. This is one of the reasons for the creation of the Ethereum blockchain. Its important to note that this paper lacked a lot of information on how the internals of Bitcoin actually worked. That said, the code base is open source, meaning that we can easily read it and understand it.

### 2.1.3 Ethereum

Basis for the Ethereum blockchain was initially explained in 2013 by Buterin et al.[5], focusing specifically on the proof-of-work algorithm used by Bitcoin and its related problems with memory-hardness, further proposing a possible alternative, which would later be used on the Ethereum blockchain. A year later, Wood et al.[23] published the paper "Ethereum: A secure decentralised generalised transaction ledger", further adding

to the main objective of the Ethereum blockchain as an attempt to build a generalised technology where anyone could develop their own decentralized applications and smart contracts.

This was a great evolution of the blockchain technology, as people no longer had to worry about developing their own blockchain, instead building their project or solution directly on top of the Ethereum blockchain. This is of great importance as we will focus primarily on Ethereum based protocols for the scope of this thesis. According to a Coindesk article [1], by 2021 there were over 3000 decentralized apps running on the Ethereum blockchain, from stablecoins to exchanges, lending and borrowing platforms, private messaging systems and many others, and most of the protocols already supported by Anchorage are built on Ethereum, such as MakerDAO [2], with others being a fork of the Ethereum blockchain, such as Celo [3]. This provides an already generalized framework to work with, as the underlying technology is the same for most protocols, with the exception of forks, which even then do not differ much from the original blockchain. On the other hand, because of this same reason, the task of developing an API that works with any Ethereum based application (either on top or fork) becomes much more doable.

It is also important to note that on September 15, 2022, the Ethereum network went through what we now call the merge, which was the process of merging the original Ethereum network (Ethereum 1) with a proof of concept network that had been in development (Ethereum 2)[4]. Because of this merge, Ethereum is no longer a proof of work based blockchain, instead being proof of stake based, and is composed by an execution layer which handles transactions and executions, and a new consensus layer, which handles the proof of stake consensus mechanisms. Although this does not directly affect our project, it is still of immense importance for the entire Ethereum project, as it effectively eliminates block mining, which in turn has dropped Ethereum's energy consumption by 99.95%, from around 112 TW/yr to as little as 0.01 TW/yr[5]. The merge also affected the scalability of the entire blockchain, by being the early steps to implement sharding, which will allow the network to be split up the blockchain itself across nodes, making it so that clients and nodes do not require to download the entire blockchain at any given moment[6]. A new Ethereum merged was also forked from this event called EthereumPoW[7], which aims at continuing using proof-of-work, although with a much smaller community.

### 2.1.4 Governance

Governance can be found on the dictionary as the activity of governing a country or controlling a company or organization, and in an economical perspective it can be defined as the use of institutions, structures of authority and collaboration to allocate resources and coordinate efforts and activities in the economy. In the blockchain space, the term Governance has been tightly linked to Decentralised Autonomous Organisations (DAO), although with no common understanding nor generally accepted formal

[1] https://www.coindesk.com/learn/2021/02/08/which-crypto-projects-are-based-on-ethereum/
[2] https://makerdao.com/en/
[3] https://celo.org/
[4] https://ethereum.org/en/upgrades/merge/
[5] https://ethereum.org/en/energy-consumption/
[6] https://ethereum.org/en/upgrades/sharding/
[7] https://ethereumpow.org/

definition [7].

Katende et al.[15] explains a DAO as a computer program that runs a distributed peer to peer network with the objective of operating without the need of any human involvement. This is done by completely removing any central governing body and giving control to the entire online community. The community can collaborate and operate the DAO by transcribing organisational processes (processes that allow the enforcement of decisions directly related to the organization) into algorithms, which run on the blockchain itself. These algorithms that represent the organisational processes are what allow for on-chain governance. Fischer et al.[7] describes governance on the scope of blockchain technology as the integration of norms and culture, the laws and code, the people and institutions that facilitate coordination and determine a given organisation. Its what allows for a decentralized organization to be fairly governed and coordinated without the need for a central body. Furthermore, because of consensus mechanisms and transparency, blockchain technology is a great enabler of governance for autonomous organisations, and in blockchains such as Ethereum, where developers can deploy their own smart contracts, the aforementioned algorithms can be implemented without the need to develop their own blockchain.

Because the main type of blockchain protocol we will be focusing on is digital currency, we can further contextualize governance as the act of deciding the future of a protocol and currency as a community through on-chain proposals and voting. During this project we focused on many different Ethereum based protocols, such as MakerDAO[8], Compound[9], Aave[10] and DYDX[11], working to implement all of these protocols in our main solution. All of these protocols allows the community to create proposals that can change anything related to the protocol itself, such as governance processes, economic processes and consensus on community goals and targets [12], and can be proposed by anyone, although in some situations it must first pass through an initial off-chain voting (MakerDAO).

These projects also supply tokens specific to each one of them, which are used for governance processes and always contains a monetary value, which can fluctuate. Institutions who invest in such currency must stay on top of these proposals, as they can determine future value of the currency itself. The most common processes in governance are proposal creation, proposal voting, and vote power delegation. In many situations, vote powered is defined by the amount of tokens of a specific protocol a user possesses, meaning that a user with more tokens will essentially have more voting power, where each token represents a vote. This is a system that as created plenty of debate, as some worry about the possible imbalance of power, as more wealthy people will have better control over the protocol, while others defend that this way people who invest more in a protocol are rewarded accordingly.

[8] https://makerdao.com/en/governance/
[9] https://compound.finance/governance
[10] https://governance.aave.com/
[11] https://dydx.community/dashboard
[12] https://makerdao.world/en/learn/governance/how-voting-works/

## 2.2 Technical Analysis

A technical analysis is extremely important when analysing blockchain solutions, as of time of writing there is a large gap when it comes to scientific material related to this topic. As such, an analysis must be performed through a combination of other means, such website research and source code analysis. In this section, we will go through the aforementioned projects, understanding how both Bitcoin and Ethereum work, how these projects work, and what functions we will need to work with in order to achieve our objectives.

### 2.2.1 Bitcoin Technical Details

Although it is common to generalize the blockchain as a series of blocks each containing multiple transactions and each chained to one another (as we previously did), it is also important to understand how a bitcoin transaction actually works, as it ties in on why Ethereum was created and how it works. More specifically, it is important to understand how these transactions are actually locked to a user and kept exclusive to that same user.

Transactions in bitcoins are done between addresses. An address is a smaller identifier obtained from a public key, which in itself is obtain from a private key, which we will better explain further on the text. Its important to note that private keys, as their name suggests, should always be maintained private, while public keys and addresses can be shared in order to create transactions. Bitcoin transactions to an address are not summed up into a single value, and are instead composed of one or more inputs and one or more outputs, where the input represents some or all of the bitcoin one might own, and the output represents who will receive a set amount. When we receive bitcoin, we are given access to the outputs of a transaction, which are called Unspent Transaction Output (UTXO), that can then be used as input to another transaction. When we want to transfer bitcoin to someone else, we then use UTXO as an input to our new transaction, and finally define new outputs for the recipients of the transaction we are creating. Furthermore, it is important to note that bitcoin UTXOs must be used as whole amounts, meaning that if we want to create a transaction that requires less bitcoins that what an UTXO has, we can not just remove the wanted value, as an UTXO must be completely used. Instead, we use said UTXO as an input, and we create one output of the desired value we want to transact, and a second output back to ourselves with the remaining unused amount, thus giving us access to a new UTXO with the difference of the original existing amount and the amount transacted.

That said, not everyone can use the outputs of any transaction they wish, as Bitcoin makes use of Elliptic Curve Digital Signature Algorithm (ECDSA) algorithms to control who owns each output of a transaction. ECDSA allows users to generate private and public key pairs, and then prove they are the owner of said public key without ever sharing the private key, instead generating a digital signature from the private key, which by means of a mathematical algorithm proves that you own the public key.

In Bitcoin, this is implemented via Script, a rather small coding language purely used to lock and unlock outputs. Its extremely basic, and only contains two simple elements, being data, such as a digital signature, and Opcodes. Opcodes can be compared with common Assembly, and consists on low level instructions executed in a stack like data structure using LIFO (Last in First out) as a data queuing method, which in turn allows

us to build simple functions, which in Bitcoin are called scripts. These scripts are a combination of opcodes and data, and are applied in two separate steps. First a locking script, also called a scriptPubKey is placed on each output of a transaction. After, when we want to unlock said output to use it as an input, we need another script, also called a scriptSig. We then combine both the scriptSig and the scriptPubKey (scriptSig must always come first), and execute the resulting function. As long as the combined function returns any other value that is not equal to zero, the output has been unlocked and can be used as an input in another transaction.

Bitcoin already has multiple standard scripts. The first standard script implemented was called Pay to Public Key (P2PK), and as the name indicates, it allows us to lock an output to someones public key, which can then be unlocked using a digital signature that can only be obtained by the holder of the private key used to generate the public key used to lock the output, making use of our aforementioned ECDSA capabilities.

With all of this information in mind, it becomes apparent that Script could be used for more than just locking and unlocking. The scripts can be more complex, allowing us to create new locking and unlocking mechanisms, and even save a certain value on the lock, which can then be used when unlocking, effectively giving us some form of value persistence, albeit very robust. In fact, some projects actually tried to build upon this idea, which believed that in order to create an application supported by blockchain, you didn't have to actually deploy your own network, but instead use an already existing network, like bitcoin, in turn using their transaction system as a wrapper for their own transactions, and Script as a way to code custom functionalities, and even store data. An example of this is Colored Coins, a project aimed at allowing people to create their own crypto coin using bitcoin as the underlying blockchain, which would tag the UTXO with colors in order to distinguish the different custom coins. Metacoins is another great example, building a custom protocol on top of the bitcoin blockchain, and just like Colored Coins, using bitcoin transactions and UTXOs to track custom coin transactions. This idea of using a single blockchain to fuel multiple different applications and coins quickly became something with a lot of value, opposing the creation of custom blockchains for very specific usages, and can be seen as an early version of what we today call Smart Contracts.

Nonetheless, its clear that this solution comes with multiple problems that can hinder the capabilities of the scripting language and its functionalities. Although it includes an extensive library of opcodes, Script is not considered Touring complete, as it still lacks a lot of support for other operations, including loops. Furthermore, Script lacks the capacity to maintain and save the current state, meaning that there is no variable persistence, only the UTXO of the next transaction. Finally, any locking script that is non-standard on the bitcoin blockchain would not be relayed from the memory pool of nodes, meaning that it is rather complicated to have them added to a block and mined, and in most cases requires a fully custom node.

### 2.2.2 Ethereum Technical Details

Although some developers started to work with what bitcoin could offer, others saw bitcoin as a proof of concept of something that had not been done before, and quickly started working on new blockchain technologies that attempted to rework what bitcoin could offer, changing trade-offs and making it better. One of the most important blockchains born from this vision was the Ethereum blockchain.

The Ethereum smart contract webpage[13] further explains the concept of smart contract as a program that exists and runs on the Ethereum blockchain. It is a collection of code (representing functions) and data (representing the state), and just like a normal Ethereum account, resides on a specific address on the blockchain.

Smart contracts can also hold an Ether balance and partake on Ether transactions. That said, once deployed to the Ethereum network, they are not controlled by a user, instead being interacted with by other user accounts by submitting transactions that execute specific functions defined on the smart contract, which can change the state and return values. Many projects based on Ethereum, which we call protocols or projects, are composed by multiple different deployed smart contracts, which users interact with, and even interact between each other to provide some sort of service.

Although Ethereum still acts as a distributed ledger for Ether, the official Ethereum technical documentation[14] further explains how the Ethereum blockchain was no longer seen as a distributed ledger, like Bitcoin, but was instead a distributed state machine. The Ethereum blockcahin, at an elemental level, is a large distributed data structure, which not only contains account data, transaction data and balances, but also machine states, which hold states and variables, and can execute arbitrary machine code. In the official Ethereum whitepaper[15], the state of said machine is further defined as being made up of objects called accounts, which are represented by 20 byte keys, contain transaction nonces, Ether balance, and contract code and storage, if the account is a smart contract account, and state transitions, which represent blockchain transactions. For our project, understanding how transactions work is extremely important, as it is one of the steps we are trying to abstract, and although we will do a more in-depth analysis on how transactions work, we should still mention the more important aspects.

In Ethereum, transactions are seen as signed data packages that can store messages. These messages can be simple Ether transactions, just like Bitcoin transactions, but can also be smart contract messages, used to communicate with smart contracts. A transaction will always contain a transaction recipient, a transaction sender signature, a certain amount of Ether to be sent (which can be zero), and two Gas related fields which we will discuss further ahead. These fields are mandatory, and always have the same format. That said, there is another optional field called DATA. This field is what allows us to communicate with smart contracts by packing data that a smart contract can read. This field, although encoded with the same method for almost every message, can have its data vary extensively, as it is directly dependent on the smart contract function parameters. Two smart contracts that share functions with equal parameter count and type will share the same data field before encoding, while two smart contracts with different function parameter count and type will have a much different data field before encoding.

We should also go back and understand what Gas is. According to the official Ethereum Gas page[16], Gas is a unit of measurement used to measure the amount of computational power required to execute specific operations on the Ethereum network, and is a fee necessary for every transaction. Gas is paid in Ethereum's native currency Ether, and can be set as a parameter by the creator of a message, allowing to change the

[13] https://ethereum.org/en/developers/docs/smart-contracts/
[14] https://ethereum.org/en/developers/docs/evm/
[15] https://ethereum.org/en/whitepaper/
[16] https://ethereum.org/en/developers/docs/gas/

`STARTGAS`, which defines how much Gas the transaction has, which in turn defines how many computational steps the transaction is allowed to make, and the `GASPRICE`, which represents the amount of Ether the sender pays per computational step. The amount of gas necessary changes depending on the operation, with the smallest computational step costing one gas unit. Further more, each byte in the DATA field costs an extra five gas. After deciding how much Gas we are willing to spend, we also need to decide how much Ether we want to spend per Gas unit. Every Ethereum block sets a base fee for each Gas unit, which is then multiplied by the amount of Gas you are willing to spend to get the maximum amount of Ether you are going to have to pay. Furthermore, on top of this base fee, users can also pay a tip per Gas unit. This tip is summed with the base fee before the multiplication, and after a transaction is mined, the total amount of the tip (tip value multiplied by the amount of gas you spent) is actually given to the miner who mined the block where your transaction was included, meaning that the amount of Ether you spend on Gas can effectively make your transactions be processed faster, as miners prefer to focus on transactions with higher tip values, so they can make more profit. Gas is also crucial for the Ethereum network, as it denies possible attackers the capability to hog network resources by creating infinite loops, as each loop will cost a certain amount of Gas an attacker must pay before hand.

Now that we understand how Gas work, we can finally understand how a decentralized state machine actually executes code. Smart contract code in the Ethereum network is handled by the Ethereum Virtual Machine (EVM). We must understand that the EVM is not a cloud virtual machine or nothing of the sort. The EVM is a distributed state machine, which changes from block to block, and can execute arbitrary code (smart contracts). The EVM runs according a set of rules, meaning that, given a certain input, the output will always be the same anywhere. As such, we can imagine how each node contains an instance of the EVM, and when a smart contract is called with a given input, the output will be the same across every node. The state of the EVM itself is a large data structure that not only contains accounts and balances, but also the machine rules to execute smart contracts and the smart contract code itself. Its important to understand that every time a transaction is sent to a smart contract, any node that is validating the transaction executes the requested smart contract function with the received data, just like it would be done if it was just a normal Ether transaction.

The Ethereum Whitepaper webpage[17] further explains how said code execution works. Smart contract code in Ethereum is written in a low-level, stack-based bytecode language, which sounds extremely similar to Bitcoin Script, where code is represented in the format of bytes. The big difference from Bitcoin Script comes in two main forms. First, the difference in available opcodes between EVM code and Script is rather large, as EVM code is considered turing complete. Second, Ethereum comes with an high level language on top of EVM code called Solidity, which resembles languages like python or javascript, and compiles directly into EVM code, making it much easier and faster to write smart contracts.

It is also important to understand how these smart contracts work memory wise. EVM data is split into three different elements, namely the stack, memory, and storage. The stack is exactly the same as the Bitcoin Script stack, also in LIFO mode, where operations and values are stored during smart contract execution. The memory is a temporary byte array of close to infinite size. And finally the storage, which is part of

[17] https://ethereum.org/en/whitepaper/

the EVM state, and stores key/value pairs, which include the smart contract variables. Its important to note that the stack and memory are reset after each computation, unlike storage.

Furthermore, smart contracts can contain many different functions that can be called, which can be accessed in different ways. Functions that change the storage of the smart contract are always accessed through an ethereum transaction, as variables are changed permanently, and as such Gas is necessary for the computation needed to change the state of the EVM. On the other hand, functions that do not change smart contract storage nor variables do not need an Ethereum transaction, and can instead be called directly through the EVM, as the state does not changes. This is usually used to get existing values. In order to call functions from the outside, such as web services and our own service, Ethereum produces an Application Binary Interface (ABI), a standard way to interact with contracts, which consists on a JSON file that exposes and describes the deployed smart contract and its functions[18]. When calling a function from a smart contract, the corresponding function ABI is encoded in order to produce the Data field of a transaction. This process is well documented in the Solidity documentation[19], where this data field is explained as being composed by two different parts. The data field starts with a selector, which is the first four bytes of the signature of the function we want to call. Here, the signature is the `Keccak-256` hash of the function and its parameter types. The second part is the argument encoding, where our input arguments for the function are located. Argument encoding is done per variable, and differs between variable type. Most variables are encoded in hex format and padded to 32 bytes. Some variables, more specifically dynamic variables, like byte array and strings, are made up of multiple groups of 32 bytes, including an offset group defining where a variable starts, a length group defining the size of the variable, and finally the variable itself. Parameter encoding is also used standalone without a selector is also used in other occasions, such as function return values, where we must decode them ourselves.

Note that EVM code does include methods to handle errors. Although this might now seem obvious in any modern coding language, in Ethereum this shows much more importance as every transaction you make costs Gas, which means that in case a smart contract fail, we can lose the Gas we paid, and yet get nothing in return. In 2017 an Ethereum proposal was presented called EIP-140: REVERT instruction[20], which aimed at implementing a new opcode that not only gives us a way to stop execution of code, but also revert any change done till that moment and without spending any gas. This opcode is used by the require() error catching function, and it means that, in case the execution of a smart contract fails due to catching an error using require, the state of the machine is reverted, but more importantly, the remaining Gas we paid and was not used is refunded to us. On the other hand, the error catching function assert() also reverts the state of the machine, but does not refund Gas, instead spending it all.

With a better understanding on how both Ethereum and the EVM works, it is now possible to better analyse some of the projects we want to support, being `Compound`, `DYDX`, `Aave` and `MakerDAO`, all of which Anchorage already supports, which in turn will allow us to better understand how both voting and delegating work on these projects.

[18] https://ethereum.org/en/developers/docs/smart-contracts/compiling
[19] https://docs.soliditylang.org/en/v0.7.0/abi-spec.html
[20] https://eips.ethereum.org/EIPS/eip-140

Note that the following analysis is done not only through online information, but also through direct source analysis that can be found on the appendices of this thesis.

### Compound

Compound[21] is an Ethereum protocol project that allows users to lend and borrow other cryptocoins (other projects) directly on the blockchain[22] while allowing users o lend to earn interest over time, and it is a project guided by governance using their native token `COMP`[23].

Compound was the first analysed governance project, and the easiest to understand, given its simplicity. In appendix B we can see how the public delegate function calls a private function. In order to delegate, we must only give an address defining who the delegatee is. We can also see a `delegateBySig` function. Although we will later talk about this type of functions on a different chapter, for now we will ignore it, as it does not concern our code development, given that it is not widely used, and does not provide any benefits. We should also note how the delegators address used in the private function is taken directly from the transaction (msg.sender), and not explicitly given as a parameter. Although for now this is not important, it will later be analysed in further detail as it can create unwanted restrictions for the development of our standard proposal.

In appendix C we can observe a similar situation, where voting is done through the `castVote` function, which in itself calls the corresponding private function. Voting only requires a proposal identifier and a Boolean representing your decision (abstaining is not supported). We also have access to the proposal propose functions used to create proposals. Although creating proposals through our service is not a requirement, we decided that it is important to understand how it works, in order to deploy and fully test the smart contract, thus allowing us to create a testing environment for our service.

### MakerDAO

MakerDAO[24] represents the Maker protocol[25], a crypto currency which uses any Ethereum based asset as a collateral, with the advantage of being a stable coin, meaning that the value of coin is not as volatile as Bitcoin or Ether itself.

MakerDAO is, however, different from any other project analysed at a technical level, and although it does contain more complex code, it was generally much more simple to test. One major difference in Maker is that, although we still use one token as voting power (MKR), we also get another token in exchange every time we vote (IOU), in order to continue voting. This is because in Maker it is necessary to lock our tokens in order to vote, which means that we can't use said tokens for anything else while they are locked. IOU tokens are given back when we lock MKR tokens in order to vote on a different type of proposals, as seen on the Lock function in appendix E, which we will not support, as Anchorage does not support them either. Furthermore, as we can see in appendix D, unlike Compound, Maker does not contain any delegate functions. Instead, in order to delegate in Maker, we must approve someone else address to vote for

[21] https://compound.finance/
[22] https://www.gemini.com/cryptopedia/what-is-compound-and-how-does-it-work
[23] https://compound.finance/governance
[24] https://makerdao.com/en/
[25] https://makerdao.com/en/whitepaper/

us, which essentially gives them access to our voting power. That said, and somewhat like what happens in Compound, if you want to vote directly instead of delegating to someone else, you still need to approve a specific address, which in Maker is the Chieftain smart contract address (appendix E), otherwise you will not be able to vote. Finally note in appendix E how the function used to vote for a proposal is called vote, unlike Compound which is called `castVote`, which can already give us a glimpse of a major problem we will face developing the service itself.

### Aave

When it comes to its utility, Aave[26] closely resembles Compound, with the main different being that it also allows the borrowing and lending of real-world assets.

Aave introduces some changes on how smart contracts are composed by using libraries. The code seen in appendix F is part of a file called `GovernancePowerDelegationERC20`, created specifically to be used by ERC20 projects. This is of extreme importance, as any project that uses this file to define their delegation functions will have the same code, and thus facilitating abstraction. Although `GovernancePowerDelegation` introduces the concept of delegation types, this is currently ignored by most projects, and as such we can focus only on the public function `delegate(address delegatee)`, which, just like Compound, only requires an address that represents the delegatee. Voting also shares similarities with Compound as we can see in appendix F, by only requiring a proposal identifier and a decision, which can also only be true or false. That said, there is one major difference, which is the function name itself, which, unlike Compound, is called `submitVote`.

### DYDX

Finally, DYDX[27] does differ a bit from all the other projects analysed, as its main purpose is to be a decetralized exchange for cryptocurrency trading, with most of the trading projects residing on the Ethereum blockchain.

DYDX is extremely similar to Aave when it comes to Governance matters. Looking at appendix H we can see how delegation is exactly the same as Aave, as they share the same source file. In appendix I we can also see that, even although the file is not the same, the voting functions is essentially the same, sharing both the function name and required input parameters.

### Signature Recovery

It is also important to note that both Aave, DYDX and Compound include extended voting functions with the suffix called `BySig`. These functions differ from the normal functions on the address the function will affect. Generally, when we cast a vote on a smart contract, the vote is cast for the address returned by `msg.sender`. Some times this might not be the behavior we are looking for, for example when we want to cast a vote for a specific address, but sign the transaction with a different address, which can be seen as a way of delegating votes, and signature verification based functions allows us to do exactly that. Signature verification allows us to obtain and verify an address based on a set of input parameters and a preset digest that both the user and the smart

[26] https://aave.com/
[27] https://dydx.exchange/

contract know. It utilizes a combination of ECDSA and `Keccak256` in order to both create a digest and sign with a private key, and later recover the corresponding address using the same digest and the previously created signature.

These functions, on top of the already existing input parameters, further accept at least three new variables, being `uint8 v`, `bytes32 r` and `bytes32 s`, and all of these variables are obtained by first hashing a pre-determined digest using `Keccak256`, which is then used with the users private key to create a signed message. This signed message is then split, where the first 32 bytes are `r` and the following 32 bytes are `s`. The last byte is either a `0` or a `1`, but because Solidity's implements ECDSA with some differences, this last byte must be converted into either `27` or `28` respectively, which is the value of our `v` variable.

Once we have these three variables and send a transaction, the smart contract will then use the function `ecrecover(digest, v, r, s)`, where the digest is the same as the one we used to create our signature, and the `v`, `r` and `s` variables are the ones we obtained from our signature too. The function will then return an address, which if everything was correct, will be the address corresponding to the private key we initially used, even if we used a different address to send the transaction. Although it is uncommon to use as of right now, it still exists in plenty of projects, and as such we believe that showing that it is possible to execute this type of functions through our services is important, and will also implement an example on how to use this function type on our service.

Note that as of August 10, 2022, a signature malleability vulnerability was found in `ECDSA.recover` and `ECDSA.tryRecover` functions[28]. However this vulnerability only affects contracts that implement compact signatures defined in EIP-2098[29], where the function accepts a single bytes array, instead of the more common `r`, `s` and `v` arguments. As such, this vulnerability does **not** affect any of our implemented contracts, as neither of them implement said EIP-2098.

## 2.3 Technological

With a project that requires multiple different technologies in order to achieve the defined goals, its important to further analyse and study the chosen technologies, and to better understand their purpose, the benefits they can bring, and how they compare to other possible technologies.

### 2.3.1 GraphQL

GraphQL is a framework developed by google and publicly released in 2016, although internal usage of the same framework can be traced back as far as three years before. GraphQL introduces a new type of Web-based data interface as a possible alternative to the now standard REST-based interfaces, and implements a schema to describe the organization of the data and a declarative query language for data access [10]. It is becoming increasingly popular since it was first released, as it offers a multitude of benefits and advantages over REST which we will further explain on this chapter.

[28] https://github.com/OpenZeppelin/openzeppelin-contracts/security/advisories/GHSA-4h98-2769-gh6h

[29] https://eips.ethereum.org/EIPS/eip-2098

As a web-based data interface, one of the most important and compelling benefit over REST is the possible performance gains in certain conditions, as GraphQL gives the possibility to replace multiple REST requests with a single call [10]. It is established across three different papers on the subject that, when dealing with small structured queries with small portions of data, the performance gain is noticeable, with Seabra et al. [21] obtaining a performance increase of 66%, and Jiang et al.[11] obtaining an increase of 51%. That said, GraphQL might not be the solution in every situation, as Mikula et al.[16] further analyses, noting that performance differences between GraphQL and Rest when adding, editing or deleting data was negligible, and all three papers getting decreased performance when dealing with larger amounts of data transfer. These results are nonetheless in line with the GraphQL webpage, which contextualizes GraphQL as capable of getting the exact data needed, instead of getting extra unnecessary data resulting from hardcoded and inflexible REST interfaces [8].

GraphQL implementation was also studied by Brito et al.[4], with a focus on the effort and time needed to implement a GraphQL api on a service. The study was done via an experiment, where 22 university students were asked to implement eight different queries using both GraphQL and REST. Results show a reduction of effort needed to implement a GraphQL API, with a reduction of 9 to 6 minutes, further observing that both REST experts and participants with no prior GraphQL knowledge found it easier to implement such queries in GraphQL. It was also observed an higher difficulty of REST implementation when dealing with an increase number of parameters to be used. This decrease in effort and time was finally related to the GraphQL tool provided (GraphiQL) due to a lot of its features, such as auto complete, and also because of the better syntax which allows to more easily understand code and specify parameters.

### 2.3.2 Golang

Golang (also known as Go), is a relatively new programming language developed by Google with the objective of facilitating the construction of backend software services, and targeting the world of cloud computing. Before public release, it was already used by Google itself to develop their own services. It was built from the ground up to provide comparable performance to languages like C and C#, all while implementing a relatively simple syntax that resembles dynamic languages such as JavaScript. Go programs are compiled into a single binary, and provide the ability to chose what host OS it will be ran on before compiling it, giving it excellent cross-compatibility and extremely small binary executable, which also makes Go a perfect language for microservice architectures [2].

A microservice architecture is an approach that builds the concept of software modularization, where a common service is split into smaller modules (microservices) accessible by a network interface (such as a REST API), better dividing the responsibility of each module [13]. In turn, this allows for greater scalability, only being necessary deploying new specific modules depending on the current necessities, instead of deploying the entire service again, which also provides better and more cost-efficient performance. Furthermore, there are already multiple blockchain focused libraries available for Golang. We will be using Geth[30], which is one of the original implementation of Ethereum ,

[30] https://geth.ethereum.org/

fully written in Golang itself, giving us access to every function we need to communicate with the blockchain, such as connecting to the network, crafting and sending transactions, and even working with ECDSA algorithms.

### 2.3.3 Ganache

In order to test our code, we need to setup our own copy of all the smart contracts each protocol uses, and for this we need our own private Ethereum development blockchain where we can deploy said contracts. For this we use Ganache[31], a Truffle Suit based application that allows us to do just that. We can quickly create new Ethereum networks, with Ethereum accounts included and with plenty of Ether to test transactions with. We can do simple Ether transactions from the app itself, and althgouh it also supports smart contract interactions, after a while this feature proved to be lacking in terms of functionalities and control, which lead to a necessity to use a different software. Nonetheless, through out the entire development of this thesis, Ganache was our only tool used to create test Ethereum blockchains.

### 2.3.4 Solidity

As mentioned previously, EVM code is much better than Bitcoin Script when it comes to functionality and features, but it still had a rather important drawback. The code itself was still opcode based, which, just like assembly, is extremely counter-intuitive, and hard to both write and read. As such, a high level language was created in order to fix this issue, named Solidity[32], which according to the official github page is a statically-typed curly braces programming language, much like javascript and java. It was designed specifically to develop Ethereum smart contracts, and compiles directly to EVM code, which makes it much easier and intuitive to develop code which is easier to both write and read. For this we also use an IDE called Remix[33], which not only allows us to write and test code on a sandbox, it also allows us to choose between the many different versions of Solidity available (as many different protocol smart contracts are written on different versions based on time of development), but also deploy them directly to our Ganache blockchain, interact with said smart contracts, and even debug them. Remix is not only available offline as an application, but is also available as an online IDE, which although makes it harder to connect to private Ethereum networks, it is still great to develop anywhere at any time.

### 2.3.5 Remote Procedure Call

In microservice architectures, inter-service communication is extremely important, as each module depend on each other to supply the final service. Communication between modules must be fast, steady and with minimal down-time, and has been drastically evolving over time, and can be separated between two different techniques. Remote procedure call (RPC) is a rather aged technique, where clients can remotely invoke a function just like they call a local method. The main benefit from RPC's are the fact that communication is done through binary, but most conventional RPC implementations are overwhelmingly complex, as they are built directly on top of communication protocols, such as TCP. On the other hand, REST was once most used architecture for

[31] https://trufflesuite.com/ganache/
[32] https://github.com/ethereum/solidity
[33] https://remix-project.org/

microservice related communications, as it is built directly on top of HTTP. Nonetheless, this does create further problems, as the protocol is still HTTP 1.x, which means that the protocol is text-based transport, usually in JSON format. The specific problem here is the fact that data sent in text-based transport starts as binary data, that must be converted into textual data before sending, and must be converted back into binary data after receiving. For services that do not explicitly need textual data, the conversion process becomes unnecessary.

To fix the problems mentioned, both from RPC and REST, gRPC was created, originally based from Stubby. gRPC was released by google in 2015 as an open source RPC Framework, being standardized, general purpose and cross platform. The main objective behind gRPC was to provide the same scalability and performance as Stubby did, but at a public level, Stubby being the RPC framework Google was already using before to handle tens of billions of requests. gRPC comes with a large set of features, such as communication efficiency, solving the REST/JSON text based communication by using a buffer-based binary protocol implemented on top of HTTP/2, with a simple and well-defined interface and schema, designed to work with multiple programming languages, and providing duplex streaming capabilities, further solving typical RPC one way streaming.

## 2.4 Research Methodology

As it was mentioned before, the chosen research methodology for this project is called Design Science Research Methodology, and as the name implies, its based on a design science paradigm. This paradigm is explained by Hevner et al.[12] as a problem solving paradigm, that seeks to create innovations which defines the ideas, practices, technical capabilities and products through the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished, and is further contextualized in the software development area by Cole et al.[6] as method concerned with the construction and evaluation of technological artifacts to meet organizational needs, which makes this methodology utility oriented.
This methodology shares many similarities with an action research approach, as noted by Peffers et al.[20], who further quotes Järvinen et al.[14], suggesting that, even with different terminologies and guidelines, its not possible to clearly differentiate between both methodologies. Nonetheless, the two main decision factors on choosing Design Science were the methodology focus on developing new artifacts based on organizational needs, and the supplied guidelines which can further assist us on the structure of this project, presented in Figure 2.1.

From Figure 2.1 we further identify certain guidelines (named Criterion), that will have great importance on the development of this project. "Design as an artifact" is one of the criterion we identify as important, given that the objective of this project is not only to develop a solution for a company, but rather a solution model which can be applied in different ways and contexts inside the blockchain world. "Design Evaluation" is considered of importance too, as the project is originated out of necessity for a better method which can help reduce both time and cost of development. We will further focus on "Research Contributions" by writing a well structured State of the Art that compiles and organizes the history and recent developments of Governance in a case study style section, as it is a part of the blockchain technology which still lacks research.

FIGURE 2.1: Design Research Criteria ([6])

| Criterion | Description |
|---|---|
| 1. Design as an artifact | Design research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation. |
| 2. Problem Relevance | The object of design research is to develop technology-based solutions to important and relevant business problems. |
| 3. Design Evaluation | The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation plans. |
| 4. Research Contributions | Effective design research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies. |
| 5. Research Rigor | Design research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact. |
| 6. Design as a search process | The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment. |
| 7. Communication of Research | Design research must be presented effectively both to technology-oriented as well as management-oriented audiences. |

## 2.5 Summary

With a complete analysis on the currently existing state of the art literature, we now better understand the reason why blockchain technology has grown so much recently, and how newer implementations of this same technology gives us a better way to develop our project, and allowing us to better understand the usage of governance. We further analysed the current technologies to be used, focusing on experiments that provides us with factual results, supplying us concrete evidence on why the technologies are better than other mentioned technologies.

# Chapter 3

# Analysis

In this chapter, a business value and analysis is performed, with the objective of further contextualizing the project on the blockchain and cryptocurrency investment market. We first identify and analyse the opportunity we are leveraging, followed by a market analysis on various cryptocurrencies accepted by Anchorage, and finally analysing the value of the project from both the business and customer perspective.

## 3.1 Business Value

Although value is generally associated with money or currency, more often that not that same association is a generalized conclusion, as value can come in other forms, which in the end generate a certain monetary value for the business. In this project, the concept of value is not only associated to the monetary gain one can obtain, but rather associated to the reduction of time and man power necessary to carry out a specific task, which only translates into monetary improvement if successful. Its important to understand this value, as it allows us to better define the goals we need to achieve to create a successful service. With a growing number of companies investing on crypto assets and blockchain technology, Anchorage services will become more sought after, as it enables better organization, planning and control over all their assets. That said, new blockchains and protocols have also been emerging regularly, with most of them presenting new and different features, value propositions and investment scenarios. Different companies may find new value in some of these new protocols, and as such, it is of extreme importance that Anchorage is able to keep up by supporting the governance aspect of these same protocols as quick as possible and with seamless integration, both on customer demand and to keep ahead of the competition.

Considering direct competitors to be organizations that supply cryptocurrency custody services, we have identified multiple direct competitors that offer the same service as Anchorage through web research. Although the list is rather large, there is a lack of information on which organizations actually support governance along with their service, which protocols they support governance for, and how long does it take to support governance for new protocols. We present some of the organizations that supply custody services, along with some important information in Figure 3.1.

With Figure 3.1 in mind, we can see that from 10 different organizations, only two organizations mention Governance services (Gemini, and Coinbase), and from those two only one lists the amount of currencies to which they offer governance services. Although this does not confirm how many organizations provide governance services,

Table 3.1: Competition Organization List

| Organization | Supports Governance | Protocol Count |
|---|---|---|
| Anchorage | Yes | >3 (TBC) |
| Coinbase | Yes | 1 |
| Gemini | Yes | Unknown |
| Paxos | No Mention | Unknown |
| NYDig | No Mention | Unknown |
| Circle | No Mention | Unknown |
| Bakkt | No Mention | Unknown |
| BitGo | No Mention | Unknown |
| Blockchain.com | No Mention | Unknown |
| CustoDigit | No Mention | Unknown |

it does allows us to conclude that governance services are not considered as important, thus not being a primary focus of these organizations, which in turn will give us a clear advantage on the current market. Nonetheless, because of this lack of information, it becomes much harder to do a direct market comparison and understand how far or ahead the competition really is. This also leads to a situation where analysis tools such as Analytic Hierarchy Process (AHP) and Quality Function Deployment (QFD) cannot be applied on, with no clear information available to define our decision criteria. Even so, a proper market analysis must still be done, and for this we will focus on the cryptocurrency market, as this is the market on which all of these companies rely on for customers.

To further identify the market value of our project, we can also analyse the market change of different cryptocurrencies. Even though our project is about governance, most institutions first search for an organization that provides custodian services, with governance services being an additional feature to this service. Custodian services are cryptocurrency storage solutions for institutions that hold, trade and invest cryptocurrencies, and their main focus is providing a centralized and secure storage for high-value assets. Many of the analysed organizations custodian service, including Anchorage, provide security by the means of specialized hardware to store wallets (cold storage), insurance for asset protection, and financial and organizational control. As such, we can conclude that the more valuable different cryptocurrencies get, the bigger the search for custody services by institutions who have an interest on those cryptocurrencies will grow, allowing us to define a growth factor of the current market by market price.

In Figure 3.2 we present the current market capitalization (as of time of writing) for a combination of cryptocurrency accepted by Anchorage and other famous coins, allowing us to understand how big the cryptocurrency market really is. In this context, the market capitalization of each currency is calculated by multiplying the current price of each coin by the existing coin supply.

We can further understand the possible growth of a specific cryptocurrency by analysing the price chart of the coin Ether during the last 3 years, presented in Figure 3.1.

This analysis is extremely important as it not only shows us the possible value growth and the total value of a currency, but also how quickly the price of Ether went up

TABLE 3.2: Current Crypto Market Cap

| Currency | Identifier | Market Cap. | Supply | All time high |
|---|---|---|---|---|
| Bitcoin | BTC | €749B | 19M | 60 102,94 € |
| Ethereum | ETH | €338B | 119,5M | 4 273,98 € |
| Tether | USDT | €68,3B | 78B | 1,07 € |
| Solana | SOL | €31B | 317,5M | 226,86 € |
| Maker | MKR | €1,9B | 9773,6K | 5 541,83 € |
| Celo | CGLD | €1,2B | 407,5M | 9,30 € |

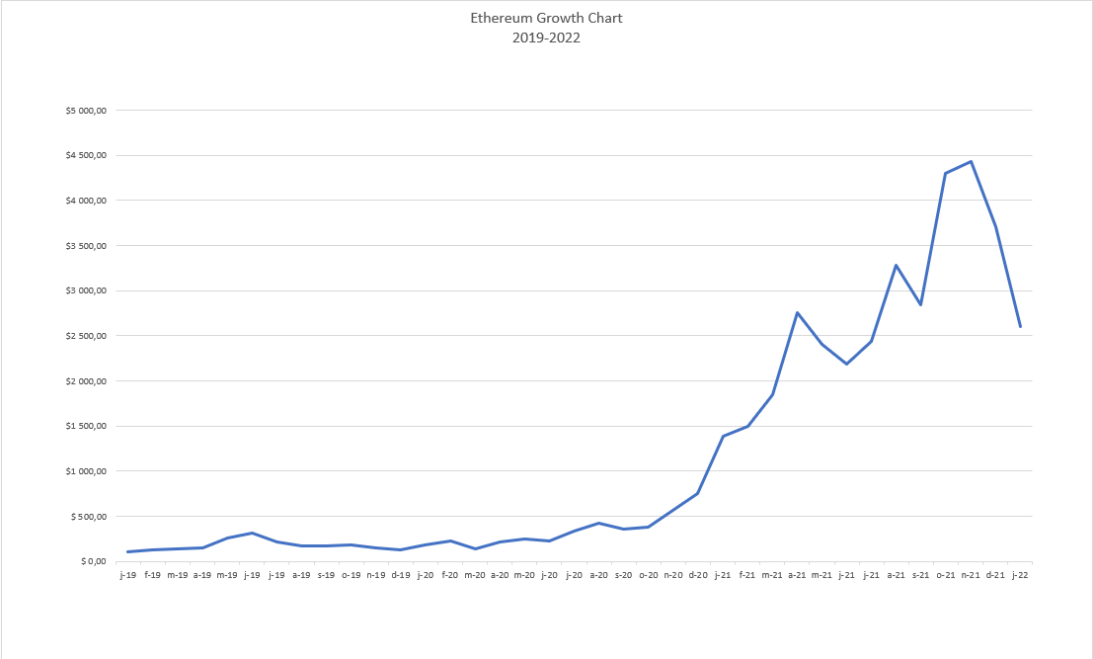


FIGURE 3.1: Ethereum Growth Chart

and how valuable it is. This is of great importance as many of the protocols we will be working to support on our abstraction are developed on top of the Ethereum blockchain, the blockchain underneath the Ether coin.

### 3.1.1 Business Perspective

As the main objective of this project is to develop a drop-in abstracted API that can be deployed on any new protocol with minimal configuration or modification, the business value is fully centered around time, manpower and cost reduction, making the necessary team size much smaller, and the development time much shorter, thus allowing for more productive employees, and less time allocation. Further more, the combination of both cost and time reduction gives Anchorage an advantage over other organizations that might not be able to keep up with the protocol support count and demand.

### 3.1.2 Customer Perspective

With the last section in mind, we will be able to improve costumer relations, by being able to support new protocols on demand, with costumers having to wait less time overall.

### 3.1.3 Value Proposition

A value proposition is a statement through which we want to demonstrate what our service provides, and how the costumer can benefit from said service. Our value proposition is based on our product value, which focus on time, cost and manpower reduction. As such, we identified the following benefits:

Development time reduction: With a fully abstracted API, the time needed to support new protocols is reduced, allowing for more protocol support faster, and more time to develop other aspects of the business;

Development manpower reduction: The need for a complete team will also be reduced, as any development necessary will be minimal, if none, thus allowing for better employee allocation on priority projects;

Standardized solution: With the same abstraction on every protocol, a API standard is created. In turn this allows for streamlined maintenance, development and bug fixing, with code changes and new features easily integrated with any already existing API;

Scalability: A fully abstracted API will also provide a new level of scalability, as new protocols can be supported on demand, and new API services can be deployed and managed with the same micro service manager, making use of a streamlined configuration.

### 3.1.4 Function Analysis System Technique (FAST)

Function Analysis System Technique (FAST), although introduced 1965, it is still to this day one of the main tools used for value management and analysis, and it provides a graphical representation of how functions are linked, in a product or process, to deliver the intended service, and mainly focus on the how and why of each step of the process. As such, the main focus of the diagram is each function, leading to a better understanding of the problem and how to better reach a solution [John2011].

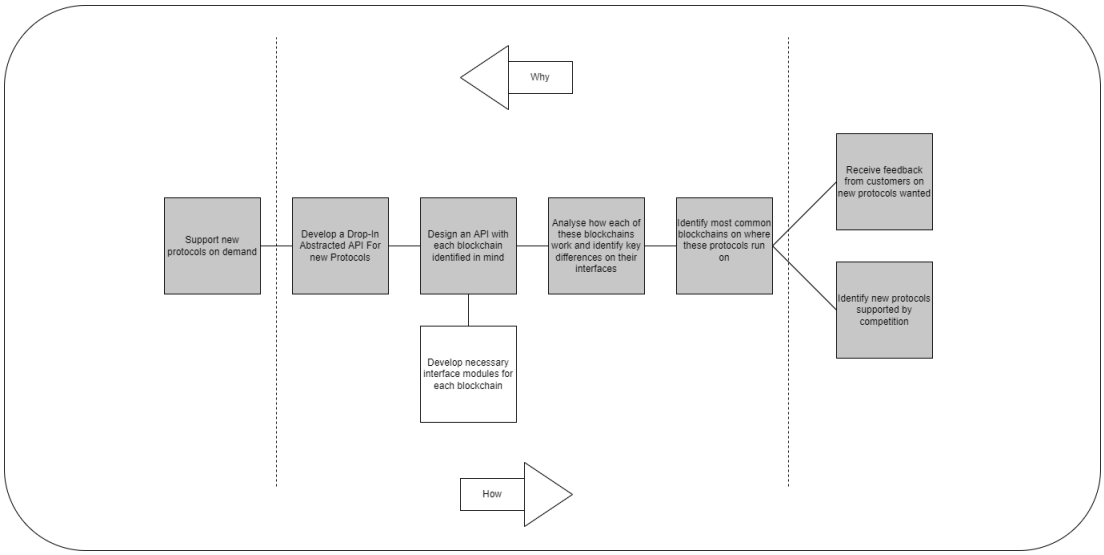


Figure 3.2: FAST Diagram

As shown by Figure 3.2, our causative elements (Lowest order element) are both receiving feedback from a customer or identifying new protocols ahead of time. These are the very base of our project, as they allow us to achieve our objective of supporting new protocols on demand. The identification and analysis of these protocols are also important steps during development, as we must first identify the blockchain on which each protocol is built on top of, and how the protocol communicates with the blockchain. The communication between the API and the protocol itself will be done through modules. This is because even after developing the API itself, it is impossible to account to every possible protocol that can be developed in the future, and as such creating a module based protocol communication will allow the API to be future proof, while also allowing for easier maintenance, organization and further development.

## 3.2 Requirements Engineering

In this chapter we analyse at a more technical level the objective of this project, including the purpose, main functionalities and requirements of the final product, through Requirement Engineer techniques. We will follow the general organization of a Software Requirements Specification document, and will start by explaining what elicitation is, what was it used for, and what results we obtained. We will the further document the project and it is multiple features and requirements at a more technical level.

### 3.2.1 Elicitation

Elicitation is an extremely important step on requirements engineering, as it allows us to better identify the various problems to be fixed and necessary functionalities for a product from those directly associated with the project itself.

For this project, elicitation was done through both interviews and system interface analysis. Interviews were conducted with an Anchorage coordinator, and were mainly targeted at better understanding the expected functionalities of the project, such as what blockchains to work with, what protocols are already in use, and what is expected as a final product. By analysing said functionalities, we further identified the necessary requirements for each functionality. A system interface analysis was also done, with the objective of further understanding how the project will fit inside the already existing environment, such as how it should communicate with already existing services, the basic design that other services expect, and which technology to use for each integration.

With all the gathered data, we then developed a specification that should fit the organization's needs and allow for the proper implementation of the analysed features. The specification further explains the purpose of the project, and delves into the technical details of the final product, such as system interfaces, functionalities and functions, functional and non functional requirements, design constraints and limitations.

### 3.2.2 Project Purpose and Scope

The main purpose of this project is to streamline the support process for governance on new Ethereum based protocols, by developing an abstract API micro service that will serve a back end service that allows users to vote on proposals on supported protocols. This project is being developed with the objective of easing the job that is implementing

and supporting new protocols and their respective governance and proposal voting, by developing a standardized API that is well documented and uniform and can be deployed on new protocols when needed, without having to make changes to the services that require said API.

### 3.2.3 Project Features

As an API, the final project can be observed as two different components. We will have one component be a GraphQL based interface, which allows other services to make requests to our API, and it can not change across different protocols. The second element can be seen as the core of our product, as it will perform the logic behind blockchain proposal voting, and will have to change depending on the protocol it will be deployed on, although modifying the code should be minimal and controlled. Features can be divided in two different perspectives. The most important feature of the product for the client will be allowing the end user to vote for proposals of their choice. For the company and the developers, the most important feature is a service that can be deployed without having to worry about the web interface, authentication and core logic, only having to modify small amounts of code that would handle the new protocol itself, and the web interface must be well documented so that it is easy to read, understand, implement, maintain and further develop. These features will be translated in both functional and non-functional requirements in section 3.2.4 and section 3.2.5. For non-functional requirements, we use a technique named FURPS+ [1], which is used to validate important requirements for a project based on necessity, and stands for Functionality, Usability, Reliability, Performance, and Supportability, with the "plus" sign allowing for method expansion by further considering constraints.

### 3.2.4 Functional Requirements

As previously mentioned, the project can be observed from the end-user perspective and the abstraction perspective, and the main functional requirements where defined as:

**FR1)** A service that must be able to vote on existing proposals on different protocols, requiring the user only to choose what he wants to vote on, meaning it must be able to receive an user id, some form of identification for the proposal on which he is voting and the vote decision.

**FR2)** A service that must be able to delegate voting power on different protocols, requiring the user only to choose a delegatee address, meaning it must be able to receive an user id and an address;

**FR3)** A service that defines a standard for this type of service with a well developed and documented interface;

**FR4)** A service that allows for deployment on new and already existing services without having to change the web interface or it is functionalities;

**FR5)** Other services that want to use our interface must be assured that the interface itself doesn't change, so that they can all use the same methods for requests.

[1] https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/

### 3.2.5 Non-functional Requirements

We further defined multiple Non-Functional requirements, requirements which do not necessarily add nor allow for specific service functionalities, but are of extreme importance as the backbone of the service itself, such as:

**NFR1)** Security: Service connections must be secure, using TLS/SSL and some form of extra encryption if necessary. Security is further assured, as all the private and sensitive data needed for the voting process, such as private keys, is stored on the HSM, and in no way is obtainable.

**NFR2)** Performance: The service must be lightweight and only do the necessary steps, and allow for high throughput by means of multi-threading.

**NFR3)** Scalability: In part a result from performance, as it allows for a easier deployment.

**NFR4)** Maintainability: Will be assured by a proper standardized and documented API.

## 3.3 Summary

With a better understanding of the current cryptocurrency and blockchain market, and the value of our project on the current market, we are able to define the requirements necessary to achieve a successful project based on both the client necessities and internal necessities, maximizing the value, which also allows us to better design and visualize the project before deployment.

# Chapter 4

# Design

In this chapter we will be further analysing different design choices on the project being developed, including the architecture of the service, how each functionality is achieved, and what technologies are planned to be used. It is important to note that, based on the fact that this project is being developed alongside Anchorage, certain architecture details might change during implementation, although it will all be noted and explained.

## 4.1 Architecture

This project can be seen as a deployable service that will supply an API for a front-end, and will have the job of converting incoming governance requests from said front end into Ethereum transactions for the respective smart contract. The objective is to facilitate the capability of proposal voting for protocol governance, and because of this it will also need to be able to communicate with the network where these protocols exist. The service must also be abstracted of multiple system elements such as databases. A diagram to better present the organization of the service and what is expected in terms of integration on the already existing environment was created, which can be seen in Figure 4.1.

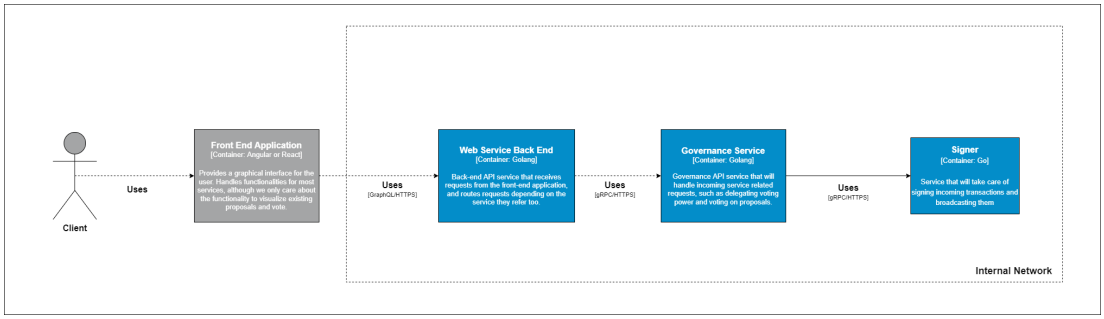


Figure 4.1: Service Architecture

From Figure 4.1 it is possible to see how an attempt is being made to abstract both the API to which requests are sent to, and the process of signing and broadcasting the transactions. The decision to abstract the service from the API was done based on the current technical environment seen at Anchorage. Front end applications are not meant to communicate directly with this service. Instead, front end applications are meant to communicate with a back-end server that, depending on the request, will re-route to another service. Furthermore, as more work was done on understanding smart contracts and the project in general, it became clear that the scope of this project

should be abstracting the interaction between services and Ethereum smart contracts, as that is where the actual problem lies on. Nonetheless,a solution that will define a standard API as originally planned will still be provided.

On the topic of transaction signing and broadcasting, it is important to understand that this process does not change specifically for Governance processes. As such, it was decided that transaction signing and broadcasting should be a separate service, which services other than this one can make use of.

The service to be developed will receive incoming requests which other services will use to communicate with, API which expect a standardized interface that does not change across different protocols. The current features (and therefore end-points) currently developed are the following:

Delegate voting power - Requires a protocol identification, user identification, and a new Ethereum address;

Vote on an existing proposal - Requires a user identification, a protocol identification, a proposal identification and a voting decision;

An end-point to obtain proposals, although originally planed, was not included because of multiple reasons. First off, Anchorage does not implement this through blockchain transactions. Although it was previously noted that it is necessary to try and abstract this service to fit multiple scenarios and not only Anchorage, it is necessary to keep in mind that Anchorage's existing system is being used to compare this solution with. Furthermore, many protocols only save an identification value for each proposal on the blockchain itself, with the name and description kept off-chain on a web server. Although at the moment this does break the ideology of decentralization, this is done because creating transactions on the blockchain with this much data would cost too much Gas, which is not ideal. This is also why Anchorage does not implement this through blockchain transactions from the get go. As such, there are two ways this can be achieved with this service:

**Obtain proposals from an API external to the company** Would mean that we are subject to the availability and/or costs of using such an API, noting also that the support of new protocols would be totally dependent on the supplier of said service. This dependency has been deemed unacceptable from a business perspective, as it would affect the ability to stay ahead of the competition. On the other hand, this method would create a more micro service oriented structure, as it would separate the process of obtaining said proposals from the service, allowing for better organization and scalability.

**Obtain proposals from an internal API** By being an internal API we are not subject to extra costs or reliability issues, also being able to support new protocols on demand. At the same time, by being a separate API from this project, it does not tie the scalability of both services together and allowing for separate development. Because of this combination, it will be the option used. Note that this method does not imply any interaction with the blockchain itself, and mainly works off the web pages available for each protocol governance proposals, which if no API is available, will require web scraping.

Do note that the necessary data can change with different implementations based on the internal architecture of a company or individual. Because of this it is only possible

to present the standard and mandatory data that is required for the service to work according to the previously defined guidelines and objectives(c.f., Section 3.2.4).

Vote power delegation means the delegator is giving someone access to the voting power, to whom is named the delegatee, so that said person can vote in their place. This is commonly used when handling multiple different Ethereum accounts, as a way to move work to someone else when the current active proposals are not of interest, or in a situation where a large number of users all look up to the ideals an individual proposes and trust said individual, thus giving him the power to vote in their name. Although plenty of organizations worry about the future of each protocol, some organizations do not share this sentiment, and as such prefer to delegate all their voting power to someone who shares the same vision they do, not having to worry about voting themselves. Here the main steps are quite simply and as followed:

1. Receive a delegation request;
2. Extract necessary data from the request, such as the protocol identifier, user identifier, delegatee identifier;
3. Build transaction and transaction data;
4. Send to signer.

Here is also important to note that a verification to check if the received user is already delegated could be done in order to prevent delegation to the same person. That said, this is usually not necessary, as the delegation process is usually far from expensive when it comes to Gas usage, and some smart contracts even implement measures to revert the transaction in case the delegatee is the same.

On the other hand, proposal voting requires a proposal ID, and a vote decision, both differing in format between smart contracts:

1. Receive a proposal vote request;
2. Extract necessary data from the request, such as the protocol identifier, user identifier, proposal identifier and vote;
3. Build transaction and transaction data;
4. Send to signer.

In this situation it is also important to note that most smart contracts require that the voting power is delegated back to the user in order to vote, which means that it is necessary to do just that in some situations.

We can see how both functionalities share pretty much the same general steps, and starts with a simple request with the necessary data. The main focus should be on how to extract the data from the request and how to build the transaction.

It is also important to focus on how private keys are handled. In order to send a transaction to the blockchain it must first be signed with a private key, which can not, at any moment, be shared or leaked, requiring large amounts of security to keep safe. It is because of this situation that this service is not attached to the signing service. The signing of transactions should be done exclusively by the organization deploying this service, as it is their job to maintain their clients data safe. In this project there is a single focus on how to build the transaction before the signature, which is still the focus

of the original objective, as signing the data does not change the process in any way, and it is not smart contract nor project dependant. Nonetheless, in order to fully test this service, a signing service was developed, which in itself is very simplistic. The servers only function is to connect to the Ethereum network, receive unsigned transactions from this Governance service, sign them, and send them to the network. Note how the signing process is the same for any possible transaction, meaning that a company does not need a signer for governance transactions and a signer for Ether transactions for example. Because an unsigned transaction (or raw) is always in the same format, one signer can sign and broadcast to the network any Ethereum transaction. Note that, in this implementation, both the main Governance service and the signing service are actually two packages in the same project instead of two separate services, in order to facilitate testing and development. Further ahead it will discuss how this would be implemented in a production scenario.

We also need to further emphasis on the fact that the signing service is nothing but a test environment service, and should not be used in a production scenario. The signing service is the service that will contain the private keys of all the clients of an organization. As such, and as it has been mentioned before, it is mandatory that this service is as secure as possible. The test service keeps private keys in plain text with no kind of protection, as they do not contain any value on the main Ethereum network and this should never be done on a production scenario. Nonetheless different methods have been designed in order to understand what the best method is.

The first method, and the easiest method to implement, would be to simply keep each private key directly on the clients device. This is unacceptable, as it would require the inclusion of each key along with a vote request, which is extremely dangerous, allowing for hackers to possibly steal the keys during transmission.

Secondly, it could be possible to simply store these private keys on a simple database, such as a MySQL database. This method is highly dangerous, as these databases do not provide enough security measures to keep these keys private and secure.

As a third option, there is also the possibility of making such votes directly from an organization owned wallet/private key, which in turn would make necessary the transfer of assets between the client and the organization, which is also unacceptable, as the organization does not and should not own any of the mentioned assets, only providing safe custody.

Finally, as a final option is the usage of an Hardware Security Module (HSM). An HSM is a hardware device, in this case in the form of a server, with the purpose of securing cryptographic processes by generating, protecting and managing keys used for encryption and decryption, and creating digital signatures and certificates, by fully automating these same purposes without the need for human intervention, thus being a closed system, with the only access possible being through a message queue and a restricted set of rules and possible commands [1]. In this case, the HSM itself not only takes care of storing the private keys, but also receives, signs and sends transactions to be broadcasted. Although it is not necessary to go into deeper detail on HSM's as they are not part of the scope of this project, it is highly advised the usage of one as the primary storage of cryptographic keys.

[1] https://www.entrust.com/resources/hsm/faq/what-are-hardware-security-modules

## 4.2 Feature Analysis

Because the objective is to create a service that facilitates the support process of new protocols, it is necessary to focus on what is required to add said protocols. In Chapter 2 it has been already explained how each protocol is composed by multiple smart contracts, and each smart contract has its own address. Further more, in Section 4.1 it was also predefined what features the service must supply to the user in general. As such, in this section four different protocols will be analysed, including their necessary functions and corresponding parameters which implement both voting and delegating, in order to understand how to link the necessary features to the respective smart contract functions, and the levels of abstraction that are possible to achieve. Its important to understand that, because each project implements the desired functionalities in different ways by using different functions and sometimes a combination of functions, it is necessary to list all these functions and map them.

Starting with both Aave and DYDX, we have a best case scenario where both functionalities are exactly the same, as it can be seen in the following list:

```
1 delegate(address delegatee){}
2 submitVote(uint proposalId, bool support){}
```

Compound, much like Aave and DYDX, only requires two functions. However, the voting function in specific differs from the other two projects, as it is named differently, as we can see:

```
1 delegate(address delegatee){}
2 castVote(uint proposalId, bool support){}
```

Finally, we have MakerDAO, the project that most differs, as it not only requires two extra functions in order to achieve the required functionalities, but also uses different naming convention and parameters on shared functions, as it can be seen:

```
1 approve(address guy){}
2 lock(uint wad){}
3 vote(address[] memory yays){}
4 free(uint wad){}
```

That said, and with these projects in mind, it is clear that not every project shares the same naming convention for functions, nor uses the same input parameters for each function. Furthermore, although Compound share the same functions as Aave and DYDX, it requires extra steps in order to work, as it will always need delegation even if the user wants to vote himself. Although MakerDAO is clearly an outlier, its still important to note how different the process is when delegating, as the function name is completely different, and the fact that voting has different input parameters.

As such, the functions of focus will be:

- `vote`
- `delegate`
- `lock`
- `unlock`

With this in mind it is possible to conclude that fully abstracting the interaction with these functions is not possible, as it requires some room to modify both the function names, parameters, and the process flow itself, making sure that it is possible to chain functionalities (delegating before voting in Compound for example) while maintaining a decent level of abstraction. It must also be assured that the developed service can chain functions, as to deal with situations like the one seen with Compound, and have enough room to develop more complex functionalities, such as signature based functions.

## 4.3 Summary

With a properly defined design plan, we now have a road map of how we can go about implementing the project, while assuring that we achieve all the aforementioned requirements, both functional and non-functional, and using the technologies that can yield the best results. The development and implementation of the project itself will be further reported and analysed in great detail in Chapter 5.

# Chapter 5

# Implementation

After designing two different solutions, alongside all the requirements, functionalities and points of focus, a service in the format of a library was developed. This service can be imported in almost any situation and used as is in order to handle and route governance transaction requests. In this chapter we will analyse in depth how this service was developed, what was achieved, what was not achieved, and why. For ease of explanation, most of the following presented code will be in small snippets in order to be more digestible, although the complete code base can be found in appendix V. The chapter is divided in two separate parts, one being the core of the service, and another being support packages.

## 5.1 Technological Stack

The Golang programming language has proved to be a great coding language to develop micro-services, and as such was used as the coding language chosen to develop this solution. It provides great performance in a single binary file, allowing for easy and fast deployment whenever needed, thus conforming to the defined requirements of performance and scalability (c.f., Section 3.2.5).

Because of the extensive analysis done understanding how each smart contract works and time spent deploying each one of them, changes in project scope, development of a standard in parallel, and the fact that deploying the service to Anchorage itself was no longer an objective, a decision to forego the implementation of gRPC and GraphQL was made, as this would not change the final result of the project. Instead, the service was developed as a package that can be imported and implemented on any already existing code, meaning that it also abstracts any kind of communication used to call the desired functions. This in turn maintains some aspect of the functional requirement FR4 (cf., Section 3.2.4), but it does not fully implement it. Development of the service saw extensive usage of the Go Ethereum (GETH) package, and other packages developed for this project alone.

Code testing and validation was done using Ganache (cf., Section 2.3), by setting up a private Ethereum network. It is important to note that during development, Ganache sometimes showed problems handling errors, and offers no debugging capabilities. As such Remix (cf., Section 2.3) was also used during development, as it offers the possibility to hook onto Ganache, and as such debug any incoming transaction, including transactions created by the service.

## 5.2 Requirements

The following section will better explain not only how the service was developed, but also everything necessary to make it work. It is also important to note that, although many of the following implementation examples are taken from projects mentioned in section 4.2, the project used as a base for the development was actually a mock governance project created for the sole purpose of this project, which implements both voting and delegating. The reason why this was done is better explained in chapter 7, and because the objective of this service is to abstract governance processes in order to support any desired project, developing the service this way further emphasizes the service capabilities.

### 5.2.1 Governance Service

In this subsection an analysis of the code development of the service will be done, with the objective of understanding the process behind it, and how many design choices were implemented. It is important to note that in order to develop this service, it was necessary to develop a test project that development could be done with, as it was decided to use all the other projects previously mentioned for experimentation purposes, and the project configuration file can be found in appendix T. That said, during this chapter, multiple different projects will be used as examples, as they are real life projects that can properly test the limitations of the service.

Every governance transaction starts with a single request structure, represented in listing 5.1. It can be seen how it was necessary to create a compromise where it is necessary to re-use certain variables in order to keep the request size small and uncluttered. The variables `NetworkID` and `AssetID` are variables that, in production, will always accompany any request. That said, because this thesis is focused on the Ethereum network, the `NetworkID` never changes. Furthermore, take notice on the existence of both an `UserID` variable and a `PrivateKey` variable. This is because, when implemented on a production environment, private keys should never be sent through a request, nor even accessible, as the primary objective of this service is just to translate requests into transaction data, which is then sent to an HSM to be signed and broadcasted (cf., Chapter 4).

However, because deploying the service on a production environment was not possible, the variable `PrivateKey` was created as a development tool, in order to sign and broadcast the transactions without the need for an HSM. During development a signer was also created, which receives transactions, signs them, and broadcasts them to the network. Other variables, such as `ProposalID`, `Vote`, `Data` and `Address` are all variables used when executing Voting and Delegation related transactions, and necessary depending on the transaction. Do note that when a request is received, fields that are not used for that specific functionality can be empty.

```
type Request struct {
  NetworkID         string
  AssetID           string
  ProposalID        big.Int
  Vote              big.Int
  UserID            int64
  Type              int
  Quantity          int64
```

```
9     TransferReceiver string
10    Data             string
11    Address          string
12    PrivateKey       string
13 }
```

Listing 5.1: Request Structre.

When a request is received, the entry point is the function `HandleRequest()` seen in listing 5.2. Although this function is primarily code flow handling, it is important to note how the section that sends a transaction to the signer is handled by a `for` loop, iterating through an array (lines 25 to 42). Although this will be explained later, it is a crucial part of this service, as it allows multiple transactions to be chained and executed when smart contract function dependencies exist, while the client only has to call a single service endpoint.

```
1  func (gov govLogic) HandleRequest(request networkRequest.Request) {
2
3    InfoLogger.Println("Received a new transaction request...")
4    InfoLogger.Println("Request type is: " + strconv.Itoa(request.Type))
5
6    // Create an offline signer to handle transactions and EVM calls
7    signer := Signer.GetService(gov.Network)
8
9    // First obtain the asset struct
10   assetStruct, err := getAsset(request.AssetID, signer)
11
12   if err != nil {
13     ErrorLogger.Println(err)
14     return
15   }
16
17   // Get Asset action array
18   assetActions, err := getAssetAction(request, assetStruct)
19
20   if err != nil {
21     ErrorLogger.Println(err)
22   }
23
24   // For each Asset action in array, send to signer
25   for _, action := range assetActions {
26     InfoLogger.Println("Current data chunk: " + hex.EncodeToString(
       action.Data))
27     InfoLogger.Println("Current data target: " + action.Address)
28     rawTransaction, err := signer.CreateTransaction(Signer.Transaction{
       UserID: request.UserID, ToAddress: action.Address, GasLimit: 21000,
       Data: action.Data, FromPrivate: request.PrivateKey})
29
30     if err != nil {
31       ErrorLogger.Println(err)
32       return
33     }
34
35     InfoLogger.Println("Raw Transaction: " + rawTransaction)
36     returnValue, err := signer.SendTransaction(rawTransaction)
37     if err != nil {
38       ErrorLogger.Println(err)
39       return
```

```
40      }
41      InfoLogger.Println("Transaction Sent: " + returnValue)
42    }
43
44  }
```

Listing 5.2: Request Handling function.

Inside function `HandleRequest()` only two other functions of interest are called, being `getAsset()` and `getAssetAction()`, which in order to understand, it is necesary to first understand how configuration files for each governance project were created. A project (or asset) is defined by an interface and a `struct`, represented in listing 5.3. The `AbstractedSmartContract` interface is used to represent a single project, containing all the important data, such as network identifier and smart contract assets. This interface exposes the set of functionalities defined previously (c.f., Chapter 4), including Voting, Delegating, Locking and Unlocking.

That said, these functionalities are represented by another structure called AbstractedSmartContractAction. Executing a function from an AbstractedSmartContract is meant to return an AbstractedSmartContractAction structure, instead of executing directly a transaction, and these structures actually contain the transaction data and smart contract address which will be used when signing and broadcasting the transaction.

```
1  type AbstractedSmartContract interface {
2    Vote(Request.Request) ([]AbstractedSmartContractAction, error)
3    Delegate(Request.Request) ([]AbstractedSmartContractAction, error)
4    Lock(Request.Request) ([]AbstractedSmartContractAction, error)
5    Unlock(Request.Request) ([]AbstractedSmartContractAction, error)
6  }
7
8  type AbstractedSmartContractAction struct {
9    Data    []byte
10   Address string
11 }
```

Listing 5.3: Service asset interfaces.

We can better understand how these two elements work with each other through the example seen in listing 5.4. First a structure containing important information is created, such as the `AssetName`, `AssetID`(which will be used to identify the asset request), `AssetNetwork`(which is not used in this implementation, as this project is only focused on Ethereum), and two addresses, one used when calling a `Vote` function, and the other used when calling a `Delegate` function, as in every analysed project, these functions were in separate smart contracts from each other. A further variable is kept called `Signer`, which is used to transport the signer connection client in case making and EVM call is necessary.

For the functions themselves, Golang's capability to attach functions to structures is used, much like what is possible do with class functions in object oriented languages, observing this functionality right after the structure itself. When defining each function, the directive `action AssetTemplate` is used at the beginning in order to define that this function is attached to the action type `AssetTemplate` structure, and is called through that same structure. All of these functions always accept the same `Request`

structure showed above for consistency. Finally, note how the return value is an array of structures of the type `AbstractedSmartContractAction`. Has seen before before, this structure is the structure that will contain the transaction data and the respective smart contract address. By returning an array instead of a single action, it is possible to chain functions internally and execute multiple transactions one after the other, without the necessity to call more than one external function at a time. A great example of this can be observed in listing 5.5, where because voting also requires delegating, an additional internal function that returns the delegating structure is created. Because of this, both the public exposed delegating and voting function can call the internal delegating function, and append to the return array. As such, when voting is called, the function returns an array of actions with both the delegation transaction data and the vote transaction data. This array is later iterated through, and each action is used to create a transaction. Finally the function `NewTemplate` is a function used to build and return the project structure, and is called by the function `getAsset()`. The way configuration files work allows us to make sure both the functional requirement FR1 and FR2 are implemented, not only implementing both voting and delegating, but also allowing for different ways of implementing each function without affecting the way the function itself is called.

```
type AssetTemplate struct {
  AssetName       string
  AssetID         string
  AssetNetwork    string
  VoteAddress     string
  DelegateAddress string
  Signer          Signer.Service
}

func (action AssetTemplate) Vote(_request Request.Request) ([]
    AbstractedSmartContractAction, error) {

  return nil, errors.New("assetHandler: Asset named " + _request.AssetID
     + " does not contain Lock function")
}

func (action AssetTemplate) Delegate(_request Request.Request) ([]
    AbstractedSmartContractAction, error) {

  return nil, errors.New("assetHandler: Asset named " + _request.AssetID
     + " does not contain Lock function")
}

func (action AssetTemplate) Lock(_request Request.Request) ([]
    AbstractedSmartContractAction, error) {

  return nil, errors.New("assetHandler: Asset named " + _request.AssetID
     + " does not contain Lock function")
}

func (action AssetTemplate) Unlock(_request Request.Request) ([]
    AbstractedSmartContractAction, error) {

  return nil, errors.New("assetHandler: Asset named " + _request.AssetID
     + " does not contain Unlock function")
}

```

```
func NewAssetTemplate(signer Signer.Service) AbstractedSmartContract {
  return AssetTemplate{
    AssetName:       "AssetTemplate",
    AssetID:         "AST",
    AssetNetwork:    "ETH",
    VoteAddress:     "0x0000000000000000000000000000000000000000",
    DelegateAddress: "0x0000000000000000000000000000000000000000",
    Signer:          signer,
  }
}
```

Listing 5.4: Project configuration template.

```
func (action Compound) Vote(_request Request.Request) ([]
    AbstractedSmartContractAction, error) {

  var returnArray []AbstractedSmartContractAction

  _request.TransferReceiver = action.Signer.GetAddress(_request.
    PrivateKey)

  returnArray = append(returnArray, AbstractedSmartContractAction{Data:
    action.delegate(_request), Address: action.DelegateAddress})

  proposalParameter := ABI.FunctionParameter{Value: strconv.FormatInt(
    _request.ProposalID.Int64(), 10), ValueType: "Int"}
  supportParameter := ABI.FunctionParameter{Value: strconv.FormatInt(
    _request.Vote.Int64(), 10), ValueType: "Int"}

  data := ABI.EncodeSig("castVote(uint256,bool)", proposalParameter,
    supportParameter)

  returnArray = append(returnArray, AbstractedSmartContractAction{Data:
    data, Address: action.VoteAddress})

  return returnArray, nil
}

func (action Compound) Delegate(_request Request.Request) ([]
    AbstractedSmartContractAction, error) {

  var returnArray []AbstractedSmartContractAction

  returnArray = append(returnArray, AbstractedSmartContractAction{Data:
    action.delegate(_request), Address: action.DelegateAddress})

  return returnArray, nil
}

func (action Compound) delegate(_request Request.Request) []byte {

  delegateeParameter := ABI.FunctionParameter{Value: _request.
    TransferReceiver, ValueType: "Address"}

  data := ABI.EncodeSig("delegate(address)", delegateeParameter)

  return data
}
```

Listing 5.5: Compound vote configuration snippet.

With a better understanding of the two main elements used to represent a project in a configuration file, it is possible to better understand how the two previously mentioned functions work. The function `getAsset()` is in charge of obtaining the asset structure that gives us access to all the possible functions a specific project allows for. The function can be better analysed in listing 5.6, and it maps each project identifier to a configuration, returning a `AbstractedSmartContract`. One drawback of this function is the fact that every time support for a new project is added, it is necessary to manually add said project's identifier to the map.

```
1  func getAsset(assetID string, signer Signer.Service) (
       AbstractedSmartContract, error) {
2
3    var AssetsList map[string]AbstractedSmartContract = map[string]
       AbstractedSmartContract{
4      "Aave": NewAave(signer),
5      "CMP": NewCompound(signer),
6      "MKR": NewMaker(signer),
7      "DYDX": NewDYDX(signer),
8      "AST": NewAssetTemplate(signer),
9    }
10
11   retValue, ok := AssetsList[assetID]
12
13   if !ok {
14     return nil, errors.New("assetHandler: Action named " + assetID + "
       not found")
15   }
16
17   return retValue, nil
18 }
```

Listing 5.6: Asset Handler Function.

On the other hand, the function `getAssetAction()`, seen in listing 5.7, receives as an input parameter the original request and previously obtained asset structure, reads the request transaction type (Vote, Delegate, Lock and Unlock), and uses the structure to call the correct function, returning an array of actions to be executed as transactions. This function partially implements the functional requirement FR3, as it does implement a standard interface to interact with, but in our opinion, due to the differences each project presents, and the way chosen to develop the solution aimed at taking into account all these possible differences, the solution is not considered fully developed.

```
1  func getAssetAction(request networkRequest.Request, smartContract
       AbstractedSmartContract) ([]AbstractedSmartContractAction, error) {
2
3    var transactionStruct []AbstractedSmartContractAction
4    var err error
5
6    if request.Type == Vote {
7      transactionStruct, err = smartContract.Vote(request)
8    } else if request.Type == Delegate {
9      transactionStruct, err = smartContract.Delegate(request)
10   } else if request.Type == Lock {
11     transactionStruct, err = smartContract.Lock(request)
12   } else if request.Type == Unlock {
13     transactionStruct, err = smartContract.Unlock(request)
14   } else {
```

```
15      return nil, errors.New("assetHandler: Action of type " + strconv.
        Itoa(request.Type) + " not found")
16    }
17
18    if err != nil {
19      //log.Fatal(err)
20      return nil, err
21    }
22
23    return transactionStruct, nil
24  }
```

Listing 5.7: Asset Action Handler Function.

With all this in mind, the functional requirement FR5 was implemented, as it was possible to use single function per feature, independent on the way the project smart contract works, while being able to adapt to different projects. The non-functional requirement NFR4) (c.f., Section 3.2.5) was also achieved by using separate configuration files for each project implementation, not being necessary to modify other projects nor the service itself (except for a small map).

The non-functional requirement NFR2) (c.f., Section 3.2.5) is no longer considered necessary, as it only pertained to the existence of some sort of private key database which was necessary to communicate with.

### 5.2.2 BySig Implementation

As stated in chapter 2, although using smart contract functions that use signature verification as an input is still unconventional, it is important to support said functions, as there is no guarantee that other projects won't use signature verification based functions only. With this in mind, when implementing DYDX, more specifically its `Delegate` function, it used the signature verification function as an example, as it can be seen in listing 5.8.

```
1  //Currently using BySig function to emphasize the possibility of using
       any type of method. Using the normal delegate is much easier, as it
       can be seen on other examples
2  func (action DYDX) Delegate(_request Request.Request) ([]
       AbstractedSmartContractAction, error) {
3
4    var returnArray []AbstractedSmartContractAction
5    //var domainSeparator []byte
6    //var structHash []byte
7    var digest []byte
8    digestPrefix, _ := hex.DecodeString("1901")
9
10   // Execute Nonce call so we can be done with this as early as possible
11   // Call function currently contains way to many values, should chip it
       down, as only 4 are necesary
12   nonceValue := action.Signer.CallFunction(_request.PrivateKey, action.
       DelegateAddress, big.NewInt(0), big.NewInt(0), big.NewInt(0), hex.
       EncodeToString(ABI.EncodeSig("nonces(uint256)", ABI.FunctionParameter
       {Value: action.Signer.GetAddress(_request.PrivateKey), ValueType: "
       Address"})))
13
14   delegateeParameter := ABI.FunctionParameter{Value: _request.
       TransferReceiver, ValueType: "Address"}
```

```
15    nonceParameter := ABI.FunctionParameter{Value: nonceValue, ValueType:
        "Int"}
16    expiryParameter := ABI.FunctionParameter{Value: "250", ValueType: "Int
        "}
17
18    //Create domainSeparator hash.
19    domainSeparator := crypto.Keccak256(ABI.Encode(
20      ABI.FunctionParameter{Value: hex.EncodeToString(crypto.Keccak256([]
        byte("EIP712Domain(string name,string version,uint256 chainId,address
         verifyingContract)"))), ValueType: "Byte32"},
21      ABI.FunctionParameter{Value: hex.EncodeToString(crypto.Keccak256([]
        byte("dYdX"))), ValueType: "Byte32"},
22      ABI.FunctionParameter{Value: hex.EncodeToString(crypto.Keccak256(big
        .NewInt(1).Bytes())), ValueType: "Byte32"},
23      ABI.FunctionParameter{Value: hex.EncodeToString(crypto.Keccak256(big
        .NewInt(1).Bytes())), ValueType: "Byte32"},
24      ABI.FunctionParameter{Value: "1", ValueType: "Int"},
25      ABI.FunctionParameter{Value: action.DelegateAddress, ValueType: "
        Address"},
26    ))
27
28    //Create struct Hash
29    structHash := crypto.Keccak256(ABI.Encode(
30      ABI.FunctionParameter{Value: hex.EncodeToString(crypto.Keccak256([]
        byte("Delegate(address delegatee,uint256 nonce,uint256 expiry)"))),
        ValueType: "Byte32"},
31      delegateeParameter,
32      nonceParameter,
33      expiryParameter,
34    ))
35
36    digest = append(digest, digestPrefix...)
37    digest = append(digest, crypto.Keccak256(domainSeparator)...)
38    digest = append(digest, crypto.Keccak256(structHash)...)
39
40    signature := ABI.CreateSignature(_request.PrivateKey, hex.
        EncodeToString(digest))
41
42    vParameter := ABI.FunctionParameter{Value: hex.EncodeToString(
        signature.V), ValueType: "Int"}
43    rParameter := ABI.FunctionParameter{Value: hex.EncodeToString(
        signature.R), ValueType: "Byte32"}
44    sParameter := ABI.FunctionParameter{Value: hex.EncodeToString(
        signature.S), ValueType: "Byte32"}
45
46    data := ABI.EncodeSig("delegateBySig(address,uint256,uint256,uint8,
        bytes32,bytes32)", delegateeParameter, nonceParameter,
        expiryParameter, vParameter, rParameter, sParameter)
47
48    returnArray = append(returnArray, AbstractedSmartContractAction{Data:
        data, Address: action.DelegateAddress})
49
50    return returnArray, nil
51  }
```

Listing 5.8: DYDX BySig Delegation.

We should start by noting that, although it is possible to implement such a functionality, the resulting code is much larger, and it does take more time to implement. Furthermore, most projects that implement this type of function require a specific nonce to

build the signature digest, which is only obtainable through a EVM call, which was necessary to implement in order to proceed. Both the `Domain Separator hash` and the `Struct hash`, which together form the signature digest, could be created through an abstracted function that every project could call, as this method was well defined in EIP-712[1]. Sadly, most projects only use this EIP as a loose guideline and not as a standard, which leads to more implementation differences, which in turn creates a situation where each project as to build the digest by itself. With all that said, it is important to note that the main objective of the service, which is to receive a request and build the transaction data, and its underlying framework required no modifications whatsoever, which means that any other project where functions may differ can still be implemented.

### 5.2.3 Transaction Signing and Broadcasting

Although developing a transaction signing solution is not part of the scope of this project, it is still necessary in order to test the service. Because of this, a development signer was created, which is loaded by the service as a package, receives transactions, signs them and broadcasts them to the network, and although an in depth of the signer will not be conducted, there are still important details to note. In order to broadcast a transaction to a network, it is necessary to communicate with a network node, and in this situation, this node is Ganache. The developed signer communicates with Ganache through RPC calls documented on the official Ethereum website [2], and implements five different calls:

- `eth_sendRawTransaction`
- `eth_call`
- `eth_estimateGas`
- `net_version`
- `eth_gasPrice`

It further implements an additional Remote procedure call (RPC) call that is wrapped by GETH itself in order to be easier to use, which is `rpc.ethClt.PendingNonceAt()`. Once a transaction request is done, the signer builds a raw transaction using the provided details, including the smart contract address, previously created transaction data, a nonce obtained using `rpc.ethClt.PendingNonceAt()`, a Gas limit necessary to execute the transaction, and finally the Gas price, obtained using `eth_gasPrice`. The transaction is then broadcasted to the network using `eth_sendRawTransaction`. The RPC call `eth_call` is used to make direct function calls to the EVM which, unlike transactions, do not change the state of the EVM, with the purpose of aiding certain functions that might require other input parameters that can only be obtained through a smart contract. Like it was noted before, the development of this package was necessary only for the testing of the service, and it can be found on Appendix W.

### 5.2.4 Testing

Although integration tests are highly dependant on an already existing framework, unit tests should never be dismissed. This service allows for unit tests using Golang `testing`

[1] https://eips.ethereum.org/EIPS/eip-712
[2] https://ethereum.org/en/developers/docs/apis/json-rpc/

library, with a simple test seen in listing 5.9. This test makes use of an online tool[3] to create the smart contract transaction data before hand, which can then be compared with the resulting transaction data from the service.

```
func TestCompoundVoteRequest(t *testing.T) {
  //First we create the correct data array with the correct data using
    https://abi.hashex.org/
  var baselineArray []AbstractedSmartContractAction

  voteDataByte, _ := hex.DecodeString(
  "15373
    e3d0000000000000000000000000000000000000000000000000000000000000
  1000000000000000000000000000000000000000000000000000000000000001")
  delegateDataByte, _ := hex.DecodeString(
  "5c19a95c000000000000000000000000
  6d1c2100cf4ed8e10a0561064dc5c331a2207f21")

  baselineArray = append(baselineArray, AbstractedSmartContractAction{
    Data: delegateDataByte, Address: "0
    x491Ee16230aF6876fD67eFBf16229Fb2672E35aF"})
  baselineArray = append(baselineArray, AbstractedSmartContractAction{
    Data: voteDataByte, Address: "0
    xb9a732a1ADaC81f463f40CF013Ac9738465Aa614"})

  //Then we create a mock request
  request := networkRequest.Request{
    PrivateKey: "70
    ee715866835d91c3e3377b462dbb2227ad786538b28fcbf4860c33198e8b36",
    AssetID:    "CMP",
    Type:       0,
    ProposalID: *big.NewInt(int64(1)),
    Vote:       *big.NewInt(int64(1)),
  }

  //We can then get an asset struct
  assetStruct, _ := getAsset(request.AssetID, Signer.Service{})

  //Getting the action struct
  assetActions, _ := getAssetAction(request, assetStruct)

  //We can now compare values
  assert.Equal(t, baselineArray[0].Data, assetActions[0].Data)
  assert.Equal(t, baselineArray[0].Address, assetActions[0].Address)
  assert.Equal(t, baselineArray[1].Data, assetActions[1].Data)
  assert.Equal(t, baselineArray[1].Address, assetActions[1].Address)
}
```

Listing 5.9: Compound voting unit test.

Further note how this is something that can be expanded as necessary, even allowing the possibility of reading a file with each project identifier to be tested, accompanied by the smart contract transaction data, which would allow us to test every single project implemented. Furthermore, even if no automation is implemented, most tests are the exact same as each other, only depending on the parameters previously mentioned, and the amount of actions returned.

[3] https://abi.hashex.org/

## 5.3 Summary

With the implementation finished, it was possible to assure four out of the five functional requirements defined as needed. Thr functional requirement FR4 was not considered to be fully implemented, as although the service was indeed developed as a package that can be easily imported on any existing service that requires governance request handling, it was not developed making use of RPC nor GraphQL technology, and thus did not achieve the expected result. It was possible to implement both voting and delegating in such a way that the request of each function from the actual smart contract was abstracted, while giving examples on how it would work. It was also developed a development signer which will allow us to later validate our code.

# Chapter 6

# Governance Standard Proposal

Although our service solution achieves our objectives to an acceptable degree, a complete abstraction was not possible due to the governance projects themselves, as function names and parameters are not standardized. Thus, further research was done in an attempt to create a better solution which came in the format of a standard proposal, also known as an Ethereum Improvement Proposal (EIP). This EIP aims at developing an abstracted and extensible method to support governance processes of new and already existing tokens as fast as possible and on demand. In this chapter this EIP will be presented and analysed in more detail, understanding the reasoning behind a governance proposal standard, the design choices made, and a possible execution.

## 6.1 Reasoning

With an immensely growing number of projects (being in the form of either coins, tokens or Non-fungible token (NFT)'s) in the Ethereum ecosystem, many of these projects have started to use on-chain governance as a method to guide project development, allowing community members to partake on important project decisions, and as such shaping the project to the user expectations and needs. Although this is great for any project and community, it does raise some problems. The common user no longer tracks a single project, many times showing interest on many at once. This means that users now need to track governance processes of each project they are interested in, with no common nor shared location to monitor them all. With this in mind, and given the focus of the ecosystem which Ethereum is built upon, many Decentralized applications (Dapps) are now trying to change this by providing centralized access to as many projects as possible (including governance processes), supplying users with a single interface to track all their projects of interest, and thus solving the aforementioned issue.

Sadly, this change comes with its own set of problems and setbacks. Each project has implemented governance processes based on their objectives, needs and ideas, which as lead to very different implementations across each project. These differences can come in the form of function naming convention or function input parameters, leading to no common communication interface, and even the entire governance process flow, such as different ways to delegate and vote, or even other functionalities that not every project implements, such as MakerDAO token locking and unlocking. This difference makes it extremely hard for Dapp developers to implement each and every single project in a timely manner, because as we saw with our own implementation, even after developing a solution that abstracts the communication with smart contracts as much as possible, it is still required extra custom code to call and handle each unique function, which in

turn delays the support of each project, further worsening when we take into account how many projects are being developed and created at any moments time.

Pushing to fix this issue brings plenty of benefits, some of them related to our solution, and others related to the Ethereum ecosystem itself. First, following a governance standard will allow developers to quickly support new projects without the necessity for unique code per project, which in turn drastically reduces time and cost to develop, all the while making interactions between projects and users faster. This in turn also helps growing the communities of each project by providing an easier way to track whats going on with the project, such as current proposals and voting mechanisms, while keeping the community more engaged, thus increasing development participation. In the end projects can expect a better course of development, with a road map better aligned with the community. Furthermore, the act of discussion and planning by the community can further help advance the current state of governance by giving feedback, new ideas, possible problems, and even future proofing the standard.

The main objective of this standard is to create a possible interface composed by a preset of functions and their input parameters that allows Dapp developers for easier and faster project support. This standard is meant to only supply a common interface without forcing a specific logic flow, and should not in any way clamp, limit or define the internal governance logic of each project, aiming to facilitate the support and growth of each project while allowing for the innovation of the governance aspect of Ethereum based projects without getting in the way. A study on the possibility of developing a solution that allows projects to wrap already existing and deployed smart contracts with our standard will be further presented, by means of a proxy. In order to achieve these objectives, we must also focus on how we can develop the standard itself. Plenty of projects have delved into different governance processes, and as such we already have data we can work off in order to analyse and better understand how governance works and what the standard requires.

## 6.2 Inherited problems and design decisions

Just like the original solution, developing a standard requires both the identification of possible problems and design decisions that can either remove or attenuate said problems. In this section we will go through some of the problems we predict will impact the standard, accompanied by possible solutions.

### 6.2.1 Voting power allocation

Voting power can be implemented in many different ways. Until now, all of the analysed projects associate an users token quantity to their voting power, which means that the more tokens an user has, the more vote power he can cast on a proposal, usually in a one-to-one basis. That said, the handling of this power can differ between projects. Both Aave and DYDX share the simplest method, where as long as the user owns tokens, they can either vote directly on a proposal, or delegate their voting power to someone else. On the other hand Compound requires that the user delegates at least to himself before being able to vote, or delegate to someone else if desired, which in our opinion is unnecessary, as it both adds an extra step to the voting process and creates a function dependency that must be achieved externally either by the user or the Dapp working as a proxy. Finally, MakerDAO implements a situation similar

to Compound, but with more steps. MakerDAO requires the user to either approve someone else as a delegate, or approve Makers Chieftain contract address in order to vote directly, followed by a locking action to lock in the users tokens before actually voting. This does not only adds a function dependency to the process, but also adds an unique functionality that no other project has. It is up to us to decided which method is the best, taking into account what already exists, and finding a compromise between a solution that is not too specific and limiting, but at the same time comprehensive enough to take into account any kind of internal process.

### 6.2.2 Proposal creation

Although creating a proposal is out of the scope of our project, we believe that a governance standard that does not define an interface to create proposals would be incomplete. All of the analysed projects all create proposals the same way, accepting an array of address, an array of integer values, an array of signatures, and a byte array of call data. All of these parameters are saved on the proposal data structure, and once voting is completed and if the vote passes, the data is used, where each address from the array is called using the call data and signature array, following the integer array as parameter values, which is how the proposal is executed and enforced. MakerDAO does handle proposal creation in a much different way, where it does not actually require the creation before voting, but instead votes directly on a set of addresses. Although it is more simple, we believe creating a proposal beforehand does allow for better organization and proposal handling.

### 6.2.3 Modularity

Taking into account voting power allocation, we can see modularity as a way to achieve a broader set of functionalities. By defining a core module and additional modules separately, we give project developers a chance to build projects as simple or as complex as they need, without creating a situation where they miss certain functionalities, or a situation where they have to implement unnecessary functionalities. Furthermore we can later develop new modules that the community might deem necessary for the growth of governance processes. This can be achieved by following ERC-165[1], which is a standard method to publish and detect smart contract interfaces, thus allowing a smart contract to implement different modules, and provides a way to identify what modules are implemented by Dapp developers.

### 6.2.4 Legacy Support

We must take into account how many projects have been already deployed on the Ethereum network which implement governance processes. As we saw in chapter 2, deploying smart contracts to the network costs Gas, which can pretty quickly become a lot of money based on the size of the smart contract. Furthermore, all of these smart contracts deployed by projects contain their own storage, meaning that re-deploying smart contracts also imply the migration of storage, which is not only extremely difficult in Ethereum, but also extremely expensive. Because of this, the possibility of wrapping already existing contracts in a proxy contract that complies with our standard was an idea that we were highly interested on. After a lot of research, this was sadly not possible. Our original idea for legacy support was to develop a deployable smart

[1] https://eips.ethereum.org/EIPS/eip-165

contract that would wrap around the original project smart contract and act as a proxy, by publishing the correct governance standard interface and receiving transactions that are standard compliant, translating said transactions and re-sending them back to the legacy smart contract. This way, already existing projects and smart contracts would not need to be redeployed in order to be standard compliant, being able to work with the standard interface and the legacy interface. In order to do this, we need to understand how we identify the sender of a transaction. When a smart contract receives a transaction, it can access its sender address by using `msg.sender`. In fact, when a smart contract receives a transaction, this is the function used to know who the sender is, as having the sender address as an input parameter would be extremely dangerous. Problems arise when we try to proxy a request through an extra smart contract. When this happens, our proxy smart contract receives a transaction with the correct sender address corresponding to the user who sent the transaction. After this we would use the function `call()`, which allows smart contracts to call functions from other smart contracts. But when the proxy smart contract itself repeats the transaction to the legacy contract using the function `call()`, the `msg.sender` the legacy contract gets is actually the address of the proxy address and not the original user, meaning that any voting or delegation done would be done for the wrong address. For security reasons there is no way to send a transaction with someone else address, but Solidity does provide a possible solution for this problem. Solidity provides a function called `delegatecall()`, and at first glance it seems it might do exactly what we need, which is calling a function of another smart contract just like the `call()` function, but keeping the original sender address. Nonetheless, after many attempts, calling this function would never result on the expected behaviour. After a more technical and in depth research, we discovered that internally `delegatecall()` does not work as expected. The `delegatecall()` function does not actually call the function of a different smart contract. Instead it loads the code from the desired function onto the current smart contract, which means that the code is being executed on the wrong smart contract, and its actually affecting our proxy smart contract storage, and not our legacy smart contract storage, thus not allowing us to work as a proxy. This, again, is not a bug nor a mistake, and is actually by design[2], given that if the function actually called the desired function on the desired smart contract, it would open the opportunity for man-in-the-middle attacks. Because of this situation, our original plan to create a wrapper around already existing smart contracts was scraped.

### 6.2.5 Future Proofing

Future proofing is a result of the problem we are facing when it comes to legacy support, and although is not something necessary in an EIP, it is a decision that can help keep the standard up to date later on if needed. Matter in fact, future proofing will be achieved by implementing a proxy pattern, which in itself is done by re-using the original idea we had for legacy support. Although we are aiming at creating a standard without any kind of problem and with as little disadvantages as possible, implementing future proofing allows us to eventually update the standard and fix problems based on feedback and data that can only be obtained after extensive testing. Matter in fact, OpenZeppelin, a framework used to build smart contracts more easily, has also delved in the concept of proxy patterns as a method to update and upgrade logic contracts without the

[2]https://blockchain-academy.hs-mittweida.de/courses/solidity-coding-beginners-to-intermediate/lessons/solidity-5-calling-other-contracts-visibility-state-access/topic/delegatecall/

need to change the interface nor migrating storage. Openzeppelin implements a more complex proxy method, where it is not a one-to-one function mapping, but instead by dynamically changing the logic function to be called based on the incoming transaction, which can further work as a fallback, meaning that if we want to call a certain function on a logic contract, we do not need to expose that same function on the proxy contract. This is of extreme importance, but not implemented in our situation simply because we wanted to mainly focus on the implementation of standard interface, which although it can be done using Openzeppelin function, it is not as explicit, as it can only reside on the logical contract, without the need for said standard interface on the proxy contract. On their proxy patter website, Openzeppelin further discusses the previously mention problem of storage collisions, alongside possible solutions, such as randomized storage slot location when creating variables. Do note that future proofing will not be mandatory for our standard, but instead something highly advisable to implement.

## 6.3 Development

As we mentioned before, the development of these standard is done through different interface modules, in an attempt to give developers different levels of functionalities. This following section presents our proposal for the interfaces of different modules, including an explanation for some of our decisions.

### 6.3.1 Core Module

```
pragma solidity ^0.8.0;

interface GovernanceStandardCore {

    event ProposalCreated(address[] _targets, uint256[] _values, string
    [] _signatures, bytes[] _calldatas);

    event VoteCast(uint _proposeId, uint8 _support, string _data);

    event VoteDelegated(address _delegatee, uint256 _value);

    function propose(
    address[] memory _targets,
    uint256[] memory _values,
    string[] memory _signatures,
    bytes[] memory _calldatas
    ) external returns (uint256);

    function vote(
    uint _proposalId,
    uint8 _support,
    string memory _data
    ) external;

    function delegate(
    address _delegatee,
    uint256 _value
    ) external;
}

interface ERC165 {

```

```
32      function supportsInterface(bytes4 interfaceID) external view returns
        (bool);
33  }
```

Listing 6.1: Governance Standard Core Module

The core module was developed with the objective of providing all the necessary functions without enforcing extra unnecessary functionalities that project developers might not need. It is the bare bones minimum required for a governance contract, and includes both functions and events that should fire when their respective function is called successfully. Note that the voting function uses an integer instead of a boolean for the support parameter. This is done so that voting is not only yes or no, but also allows for a voter to abstain. The reason why we now care about users who abstain from voting is the parameter `string _data`, which is used to give a brief reasoning for a vote decision if the user decides to. It is important to understand that, as of right now, Solidity does not allow for either optional parameters nor empty parameters, meaning that in order to use this function, it will always be necessary to give some input to this specific parameter. Nonetheless, there are plans to make optional parameters a reality according to the official Solidity github[3]. Because of this we believe that it is important to take this possibility into account and implement this field right away, and for now we advise that the input should be something predefined (for example `0x0`[4]) which the function itself can handle. Function overloading was also an option, but it would require the project developer to implement both overloaded and not overloaded functions, which we believe is not acceptable for a standard. Doing this on a separate module would solve this issue, but it would be a single function, which we believe does not justify an entire module.

Also note how we directly implement the interface from ERC-165 as part of the standard, as it is mandatory to be implemented for the smart contract to be governance standard compliant. This function should return `True` for two different interface identifiers. Both of these identifiers were calculated according to the official EIP-165 documentation[5], by applying a XOR function to the selectors of all the functions on the interface, with code included in appendix J, and when used as an input parameter when calling the function `supportsInterface(bytes4 interfaceID)`, should return `True`. The first identifier is the Core module identifier, resulting in `0xdf1132f0`, and is used to identify if a governance contract is compliant with our standard core module functions. Do note that this only includes functions, and not events, meaning that in order to be fully compliant, we need to manually read the source code. The second identifier is the ERC-165 identifier, resulting in `0x01ffc9a7`, which is used to verify if the governance contract is ERC-165 compliant.

### 6.3.2 Locking module

```
1  pragma solidity ^0.8.0;
2
3  interface GovernanceStandardLocking {
4
5      event Locked(address _user, uint256 _value);
```

[3]https://github.com/ethereum/solidity/issues/232

[4]https://ethereum.stackexchange.com/questions/6773/variadic-optional-function-params-in-solidity

[5]https://eips.ethereum.org/EIPS/eip-165

```
6
7       event Unlocked(address _user, uint256 _value);
8
9       function lock(uint256 _value) external;
10
11      function unlock(uint256 _value) external;
12  }
```

Listing 6.2: Governance Standard Locking Module

Although only MakerDAO implements locking, we do believe there might be a place for such functionality in our standard. Locking and unlocking tokens before and after voting allows developers to be more confident that exploitation of voting power is not possible, all the while being able to reward users who do lock their votes as a way to award trust. Because of this, and given that only Maker implements such a function, we believe that is something that can still be better analysed and researched, and it is best to take it into account and give developers a chance to use such function, instead of dismissing it as unnecessary. Both functions `lock(uint256 _value)` and `unlock(uint256 _value)` take as an input parameter a value representing an amount of tokens. This means that the user does not need to vote with all the tokens he has, instead being able to vote with just a specific value. This interface, if implemented, should be included in the list of identifiers for ERC-165, and the resulting identifier is `0xbcde935d`.

### 6.3.3 ActionBySignature module

```
1  pragma solidity ^0.8.0;
2
3  interface GovernanceStandardActionBySignature {
4
5      function proposeBySig(
6      address[] memory _targets,
7      uint256[] memory _values,
8      string[] memory _signatures,
9      bytes[] memory _calldatas,
10     uint _nonce,
11     uint _expiry,
12     uint8 _v,
13     bytes32 _r,
14     bytes32 _s
15     ) external returns (uint256);
16
17     function voteBySig(
18     uint _proposalId,
19     uint8 _support,
20     string memory _data,
21     uint _nonce,
22     uint _expiry,
23     uint8 _v,
24     bytes32 _r,
25     bytes32 _s
26     ) external;
27
28     function delegateBySig(
29     address _delegatee,
30     uint256 _value,
31     uint _nonce,
32     uint _expiry,
```

```
33      uint8 _v,
34      bytes32 _r,
35      bytes32 _s
36      ) external;
37  }
```

Listing 6.3: Governance Standard Action By Signature Module

This module was made specifically to work with `BySig` functions, which was already talked about in Chapter 2. Note that all the functions are exactly the same as the functions presented in the core module, only adding five extra input parameters needed to verify the identity of the sender. Furthermore there are no new events, given that there would be no difference between the already created events in the core module and this one. As such, this module should make use of the events created in the core module. These type of functions have become more and more used, as it allows a single address to vote for multiple other addresses without having to delegate, which can be useful for organizations that own multiple addresses, but like voting themselves. This module, just like any other, must be included on the identifier list for ERC-165, and resulting identifier is `0x46cca5a9`.

## 6.4 Implementation

In this section it is demonstrated how our standard could be implemented in its simplest form. In appendix K we have implemented our standard using the core module without any future proofing. Note how inheriting the interfaces `GovernanceStandardCore, ERC165` forces the code to implement every necessary function, as otherwise it wont compile and throw an error. Furthermore, by inheriting the interfaces, we do not have to re-write the necessary events, as they are already declared. In appendix L and appendix M we have the same core module implementation, but this time, we applied a proxy pattern by separating the interface and storage from the logic. In appendix L we have the interface and logic of the standard. This smart contract is completely Governance standard compliant and contains no governance process logic, although it does contain all the storage. In appendix M we have the smart contract that contains all the logic. This smart contract will be used as a function library for the interface smart contract Again, notice how it is not necessary to declare the events, as we are already inheriting them from our standard interface. Because `delegatecall()` returns a boolean representing success or error, we will take further advantage of this by using a `require()` function, which asserts our success boolean with true, and reverts the execution if otherwise. When the interface smart contract receives a request, it will execute a `delegatecall()` to the logic smart contract, which in turn will load the requested function and execute it with the provided input parameters inside the interface smart contract scope. This means that, when we delegate a function call to the logic smart contract, we do not affect the logic smart contract storage, instead we affect the interface smart contract storage, which is what we want. As we mentioned before in chapter X, Ethereum smart contracts save their storage as one nearly infinite sized array with $2^{256}$ slots, where each slot can save up to 32 bytes. Variables storage is done sequentially and continually starting at slot zero, and each variable can occupy more than one slot, depending on the type of variable and its byte size in memory. Variables that occupy less than 32 bytes will have their value padded up to the 32 bytes. That said, multiple variables smaller than 32 bytes in a row can be stored in the same slot,

as long as they fit inside the 32 bytes. Furthermore, dynamic variables, such as arrays and maps, where they still maintain a slot in order just like every other variable, but the values themselves are stored elsewhere in storage. This is done because the size of a dynamic array can change, and it is achieved by applying the `keccak256` hashing algorithm in order to obtain a new location and because we are using a 256 bit hashing algorithm, it is next to impossible to hit storage collisions. That said, this system does originate certain issues when creating proxy pattern based systems. Variables that are created in storage in the storage contract must also be created in storage in the logic contract. Furthermore, all of these variables must exist on the same storage slot on both contracts. We can see in figure 6.1 that variable Var1 is found in different slots in both contracts, being found in slot 2 in the logic contract, and slot 3 in the storage contract, as both variables were created at different moments on each contract. When we create a function on the logic contract meant to modify Var1, the function actually aims at modifying the storage slot where Var1 is, which is slot 2. Problems arise when we `delegatecall()`, which means that we are loading and executing the code from the logic contract inside the storage contract scope. This leads to a situation where the function is looking to modify storage slot 2, where Var1 was located in the logic contract, but in the storage contract storage slot 2 is no longer Var1, and instead some other variable, which leads to the function modifying the wrong variable. On the other hand, in figure 6.2, we can see that variable Var1 is in the same storage slot (storage slot 3) on both contracts, as they were both created at the same point on code. In turn, a function in the logic contract coded to modify Var1 will be looking to modify whatever variable is in storage slot 3, which means that when we load and execute the function inside the storage contract scope, we will modify the variable located in the storage slot 3, being the correct slot for Var1 in the storage contract too. With this in mind, we can conclude that for now it is necessary to create variables on both contracts in the same order, so that each storage slot maps correctly.

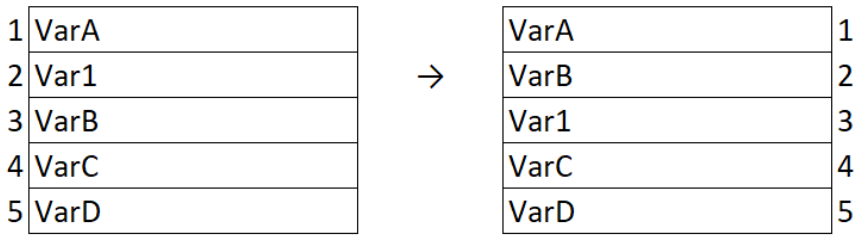


FIGURE 6.1: Proxy Storage slot collision

## 6.5 Service implementation

It is also worth nothing that this solution works directly with our prior developed service. Looking at appendix N we see how we could easily follow our template and extension example in chapter 5 in order to create the base configuration file for any project that implements our standard. Please do note that the example shown does not implement locking nor module discovery, as the original service solution was developed before the standard was finished, and was never intended to be standard focused, instead being focused on the lack of consistency that non-standard projects provide. That said, given all the examples shown, developing support for module discovery would

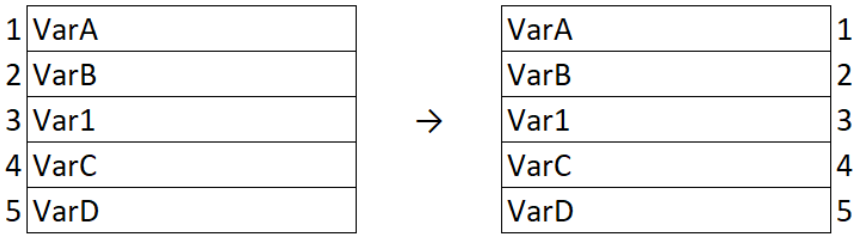


FIGURE 6.2: Proxy Storage slot matching

be completely possible, as the service itself as a framework already allows for function chaining. We can further look at appendix O to better understand how easy it is to support new governance standard compliant assets, by utilizing the aforementioned configuration file as a base, only needing to define each function address, and the asset identification.

## 6.6 Summary

We believe the information and research presented in this chapter is a solid foundation for a standard that can change how governance processes is developed in Ethereum projects, and not only achieves our main goal, but it also provides unique benefits for both developers and communities, by creating a more consistent ecosystem, and a more well defined interface.

# Chapter 7

# Experiments

With objectives and requirements defined in chapter 3 and 4, it is important to define methods to test and prove that these same requirements were properly implemented and are functional. In order to successfully implement this project, it is important to achieve the aforementioned objectives, and as such analyse the final project from an experimentation perspective. First, a set of indicators to track the various requirements previously defined will be defined, followed by a chosen methodology which will be used to analyse the indicators.

## 7.1 Indicators

Indicators will allow us to analyse and evaluate the success or failure of this project.

Based on the project itself, metrics such as code complexity and size, performance and execution time, and even code readability would be prime examples of ways to evaluate the developed solution. Sadly, because the baseline comparison is Anchorage's service, none of them are really possible, as metrics such as code complexity and readability would require access to the code base and performance and execution time are not currently measured in a meaningful way. Because of this, then main and only evaluation metric possible to compare is time to develop, which represents how long it takes to support a new project on the developed solution. This is a metric that Anchorage does support, although not in a very consistent nor accurate way. That said, it is still the most valuable metric, as it is the one that will allow to understand if the service is better than the already existing one when it comes to the main objective of abstraction, and also represents the value of the service in itself.

Furthermore, it was decided that taking into account metrics such as the number of files necessary to modify in order to support a new project, the amount of functions necessary (both external and internal) and number of lines necessary, even if no baseline is available to compare with, will help evaluate this solution by allowing to compare the different projects.

It is also important to note that the development circumstances are different between Anchorage's solution and the solution presented on this thesis. Anchorage is a rather large organization, and their development decisions were not always expecting governance to become such a requested feature. Anchorage governance services were developed as clients requested, Furthermore, the service Anchorage developed was developed alongside many other services, which impacted the way it was developed and even its priority.

## 7.2 Methodology

Due to the fact that the is only a single experimentation indicator with a baseline and three indicators without any kind of baseline, the used methodology is rather simplistic. It is first necessary to prepare the testing environment by deploying every project to our private network. After this, the process of adding support for a new project using the developed service will be timed, following a validation process where each function (Vote and Delegate) of each project is called, in order to assure that everything is working. Finally, it is correct to consider a time valid once a project passes validation, creating results that will be compared between each project.

## 7.3 Assessment

Implementation of each project was done one by one, assuring that each one works before moving on to the next project. The first implemented project was Compound, as it was also the first project analysed, and as such felt more comfortable handling. As it was noted before in section 4.2, Compound differs from other projects as voting requires that the user is delegated to himself. The method used to deal with this situation was already presented in section 5.2, and can be seen in appendix P, noting that it was neither hard nor took much longer than expected, clocking around 23 minutes of development time.

For the second implementation Aave was chosen, seen in appendix Q which unlike Compound, had no uncommon requirements for any of the functionalities, which made this the easiest project to implement, clocking around 17 minutes of development time, and no side notes.

For the third implementation MakerDAO was chosen, which presented a bit more complex implementation, requiring not only voting and delegating, but also locking and unlocking. Furthermore, because locking requires calling the delegation function, it was necessary to implement the same solution implemented in Compound, which was achieved with the code seen in appendix R. All of these factors combined did bring up the time necessary to develop up to around 36 minutes, although it did not necessarily made it any harder, simply more complex.

Finally the last implementation was DYDX, but because it shares the same implementation as Aave, it was decided to change the implementation, by using the much more complex `delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32)` function, instead of the simpler `delegate(address)` function, with the objective of both studying how much longer it would take to implement something more complex, but also to show an example on how to achieve that. This implementation, seen in appendix S was somewhat harder, as building the digest used for the signature is rather complicated and poorly documented. That said, most of the work was writing the code, which is considerably larger than any other implementation, and could be abstracted further. Nonetheless, this implementation took around 51 minutes, which is considerably longer than its counterpart Aave because of one single function.

## 7.4 Results

The experimentation results could be better if it had been possible to define better baseline values to compare with. That said, time to develop was, based on executed experiments, a success, as it can be sees in table 7.1.

Table 7.1: Experimentation analysis

| Project | Implementation Time | Files Modified | Function count | Line Count |
|---|---|---|---|---|
| Compound | 23 minutes | 2 files | 3 functions | 34 lines |
| Aave | 17 minutes | 2 files | 2 functions | 25 lines |
| MakerDAO | 36 minutes | 2 files | 5 functions | 63 lines |
| DYDX | 51 minutes | 2 files | 2 functions | 64 lines |

The times obtained were closely related to the amount of work necessary to implement all the functionalities, where Aave was the simplest implementation, with only two functions and 25 lines of code, thus showing the smallest amount of time, and DYDX being the most complex, presenting the highest amount of time. Also note how MakerDAO and DYDX, the two longest projects to implement, both share a very similar number of lines of code even although MakerDAO implements three more functions, which can be explained by the inclusion of the delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) function. Also note how for each project only two files were worked on. For each project a `Asset_XXX.go` file was created. In addition to this, for each asset the map `AssetsList` had to be updated with the asset name, in order to map the new asset to the respective function.

Again, it is important to note that, where-as we developed an entire service to facilitate and streamline the implementation of said projects, Anchorage was doing the same implementations only as needed, with a service not as developed, and a lot more testing, which was not possible to replicate given that the testing itself was highly based on integration tests with other services that Anchorage offers. In table 7.2 a comparison between our implementation time results and Anchorage project implementation time can be seen.

Table 7.2: Competition Organization List

| Project | Anchorage Implementation | New Service Implementation |
|---|---|---|
| Compound | 2 months | 23 minutes |
| Aave | 2 and an half months | 17 minutes |
| MakerDAO | 1 month | 36 minutes |
| DYDX | 11 days | 51 minutes |

That said, it is also important to note that average implementation times were further obtained based on engineer data obtained through one-on-one interviews, which result in around one to two days of code development per project, further reducing to one or two hours of code development if the code can be taken from a previously supported project. Development times obtained from development managers do not actually represent a clear picture of the development cycle. In Anchorage, just like in any other company, developing a feature requires more that just implementing the necessary code. When developing a new smart contract support, it is also required to deploy the correct smart contracts to the test network, which can be extremely time consuming when the

test network used requires the creation of a genesis block with all the necessary smart contract addresses, and develop unit and integration tests with the rest of the system and between functionalities, which in turn leads to bugs which must be fixed.

When we take into consideration how large Anchorage really is, and the considerable amount of services and systems Anchorage runs, we better understand that the time taken to implement and support a new project is not affected as much by the code writing itself, which is the main focus, but rather the deployment of new smart contracts and development and execution of tests.

This was further confirmed after discussions with some of the engineers at Anchorage, who noted that developing the code for a new project only takes around one to two hours when the project to support is comparable to an already existing project, and around a week when the project to support is either more complex than normal, or simply requires code that had not been implemented before. It was also discussed how the resulting time necessary to develop code to support said projects is not based on the complexity itself, but rather the amount of files necessary to modify, which not only requires the functionality related code, the registration of said functionality, updating lists with the new project identifier, and auto-generating files. That said, it is still possible to conclude that project implementation times fall well below Anchorage development time, while being brought down by around 98%, with an average of 32 minutes compared with the 24 hours minimum reported by some of the engineers, and 50% decrease compared to the best case scenario one hour development time. We were also able to only affect two files consistently, with an average code size of 46 lines, with no more than five necessary functions with the most complex project.

Success was achieved since implementing any project in less than a day means that we can implement multiple projects in a row before testing. Furthermore, we developed a code extremely centralized, where it is only necessary to create a configuration file and modify an already existing file, unlike Anchorage, because the code is based on configuration files found in a single place, it is also easier to follow, update and modify as necessary. The developed solution also does not require the ABI of the entire smart contract we are looking to implement, and more importantly we developed a solution that abstracts the creation of the Ethereum transaction.

It is also important to note that the developed solution further aids automatic testing, because it completely abstracts the requests from the governance projects themselves, which leads to testing becoming abstracted too, which means that for each project test cycle, it is only needed to call the predefined interface `vote()` and `delegate()`, with pre-defined requests that do not change between projects. as such, when testing each project, we do not need to create custom tests, instead only having to iterate through a list of project ID's, reusing the same two tests.

## 7.5 Summary

In this chapter we were able not only to define our performance indicators and how we were going to obtain them, but also put in practice said methodology on four different projects, testing our service based on how long it would take to support said projects. Although we faced certain problems related to the indicators we can use, we believe our results demonstrate a good level of success, as our implementation times are now

extremely small, only having to worry about a single configuration file and without having to (interact with/modify) the service itself.

# Chapter 8

# Conclusion

After months of development, multiple conclusions related to this project have been reached. It is important to note that this project has changed a lot through development, and finishing the solution design with the conclusion that the original objective could not be fully achieved was not enough for us. Because of this, instead of fully focusing on a solution that would not be enough, it was decided to split focus across two, with one solution representing what was possible to do, and another developing a possibility that would resolve the short comings of the first solution. First things first, the current governance environment in Ethereum needs some sort of standardization, or at least a community agreement to follow certain guidelines, including function naming convention and parameters, as this was the main problem that had to be worked around, and to which the standard proposal is applied to. A common naming convention and parameters would immensely help the ecosystem, and after multiple project analysis, it was also possible to conclude that in most cases it wouldn't even affect functionality itself, as most differences can be found in naming conventions. Other functionalities, such as Makers locking and unlocking, or signature based voting and delegating still require more time and analysis, as there is still not enough data to verify their true value.

Although not achieving every requirement defined at the start of the project due to multiple shifts in project scope and limitations when testing the service, a satisfactory result was still achieved, with a service that not only greatly reduces development time when implementing support for new governance projects, but also maintains the necessary configuration code much more organized and accessible, with an average development time of 32 minutes.

Finally, the prospect of an Ethereum governance standard based on the presented (cf., Chapter 6) proposal is of extreme value, as it not only creates a much more unified smart contract ecosystem, but it also aids the development of said governance projects themselves by giving to the community a much easier way to participate on the governance processes of multiple projects. It was possible to define a standard interface that includes all the necessary functionalities previously mentioned during design, creating a well defined set of function names and parameters that do not differ much from the already existing projects, nor restrict the capabilities of possible new projects being developed, with a modular system that allows users to build their projects in a building block style methodology, only implementing what they really need. The push for a proxy based smart contract layout further present a solution that will allow an inherently non upgradable technology to keep up more easily, without the necessity of re-deploying contracts every time a contract requires change.

## 8.1 Research Questions

In section 1.3 multiple research questions were defined, which were aimed at answering through out this project, being as follows:

**RQ1)** Can we abstract the main governance functionalities, such as voting and delegating?

**RQ2)** Can we develop a Framework to streamline the support of new governance projects that can be deployed on Ethereum?

**RQ3)** What limitations must we work with in order for our Framework to work, if any?

**RQ4)** How can we evaluate our Framework when adding support for new protocols in terms of development cost and time?

Starting with `RQ1)`, the answer is yes, and it was achieved through two different solutions. The first solution had to work around the limitations of the existing projects, but was able to fully implement both voting and delegating in such a way that it both allows for all the projects previously analysed, and showed the capability of supporting other projects through the configuration used.

Research question `RQ2)` was also successful, as it was not only possible to abstract both desired functionalities to an acceptable level, but it was also possible to use said functionalities to make calls and transactions to a private Ethereum network.

Research question `RQ3)` denotes the limitations of the developed service, which starts with the necessity to write custom code for each project. Although this is not necessarily originated by any of the two solutions but actually by the governance ecosystem itself, it was still something that could not be fully abstracted by creating generic functions that any project could use, and as such is a limitation.

Finally, research question `RQ4)` takes focus on the experimentation decisions previously made (cf., Chapter 7), which did not fully satisfy our objectives, as important baseline values to compare with were missing. That said, based on time to support, code complexity and modified file count, the project was successfully tested, as it was possible to lower development time compared with two different baselines, while understanding the obtained results for each experiment.

## 8.2 Contributions

Upon the completion of this thesis, three different contributions to the community surrounding the theme of this thesis were created. The focus oh a state of the art that aims at compiling as much information as possible regarding the technical aspect of both Bitcoin and Ethereum helps contributing to an area that currently lacks a lot of scientific literature. Other contributions were by supplying all the code used during the development of this thesis in order to share and show our logic and ideas, which can be built further upon. Finally, and the one deemed the most important is the proposal of a possible standard for the Ethereum community that aims at changing how governance projects are developed.

## 8.3 Limitations

The main limitation on the service solution currently is the fact that it still requires some code development in order to support a project. Changing from Golang files to YAML or XML files for the configuration of each project would further simplify the process of supporting new projects by not requiring new code, and allowing for more explicit configuration writing and reading. Furthermore, by using this style of configuration files, the usage of templates would be more effective. It is also important to note that currently, the addition of a new functionality to the service does require the modification of every implemented project, as a single interface is used for all functionalities. This could be resolved by implementing multiple interfaces inside a single main interface, which would allow to define what functionalities each project presents.

## 8.4 Future Work

When it comes to the developed service, and as it was mentioned before, it is believed that changing from Golang code to YAML or XML configuration files would bring a lot of value to the solution, as it would further reduce development time, and allow for easier template development and automatic generation.

The standard proposal, on the other hand, still needs to be presented to the Ethereum community as an EIP, which will then be analysed and discussed by community members. This will also allow for further feedback and improving of the proposal. Further exploring certain solutions presented by Openzeppelin, such as their proxy pattern implementation, should also bring more value to the proposal.

## 8.5 Personal Remarks

This was a project that since the very beginning started changing course. From taking as many existing blockchains to only focusing on Ethereum, and developing a more detailed service focused on the inner workings of smart contracts were decisions made with the objective of creating a project that instead of analysing multiple technologies superficially, would analyse a smaller subset of technologies at a much greater detail. Blockchain technology, such as Bitcoin and Ethereum, although now a rather common topic worldwide, still lacks certain aspects when it comes to scientific reading, and the main objective was to attempt at not necessarily bridging this gap alone, but rather to provide as much information and research possible around the topic of Ethereum governance, smart contracts and Golang.

# Bibliography


[1] Inc. Anchor Labs. *Anchorage Main Webpage*. 2022. URL: https://www.anchorage.com/ (visited on 01/24/2022).

[2] Mina Andrawos and Martin Helmich. *Cloud Native Programming with Golang*. 2017, p. 406. ISBN: 9781787125988.

[3] Roman Beck, Christoph Müller-Bloch, and John Leslie King. "Governance in the blockchain economy: A framework and research agenda". In: *Journal of the Association for Information Systems* 19.10 (2018), pp. 1020–1034. ISSN: 15583457. DOI: 10.17705/1jais.00518.

[4] Gleison Brito and Marco Tulio Valente. "REST vs GraphQL: A controlled experiment". In: *Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020* Dcc (2020), pp. 81–91. DOI: 10.1109/ICSA47634.2020.00016. arXiv: 2003.04761.

[5] Vitalik Buterin. *Dagger: A Memory-Hard to Compute, Memory-Easy to Verify Scrypt Alternative*. 2013. URL: http://www.hashcash.org/papers/dagger.html (visited on 02/15/2022).

[6] Robert Cole et al. "Being Proactive: Where Action Research meets Design Research". In: (2005), pp. 1–21.

[7] Aron Fischer and María Cruz Valiente. "Blockchain governance". In: *Internet Policy Review* 10.2 (2021), pp. 1–10. ISSN: 21976775. DOI: 10.14763/2021.2.1554.

[8] GraphQL Foundation. *GraphQL Webpage*. 2022. URL: https://graphql.org/ (visited on 02/15/2022).

[9] Danilo Gligoroski. "Cryptographic hash functions". In: *A Multidisciplinary Introduction to Information Security* 5.4 (2011), pp. 49–72. DOI: 10.1587/essfr.4.57.

[10] Olaf Hartig and Jorge Pérez. "Semantics and Complexity of GraphQL". In: *The Web Conference 2018 - Proceedings of the World Wide Web Conference, WWW 2018* (2018), pp. 1155–1164. DOI: 10.1145/3178876.3186014.

[11] Helgason. "Performance analysis of web servers". In: *American Society of Mechanical Engineers, Petroleum Division (Publication) PD* (2017).

[12] Alan R Hevner et al. "Design Science In Information System Research". In: *MIS Quarterly* 28.1 (2004), pp. 75–105. URL: http://www.jstor.org/stable/25148625.

[13] Pooyan Jamshidi et al. "Microservices: The journey so far and challenges ahead". In: *IEEE Software* 35.3 (2018), pp. 24–35. ISSN: 07407459. DOI: 10.1109/MS.2018.2141039.

[14] Pertti Järvinen. "Action research is similar to design science". In: *Quality and Quantity* 41.1 (2007), pp. 37–54. ISSN: 00335177. DOI: 10.1007/s11135-005-5427-1.

[15] Melina Mutambaie Katende and Rachel O Dwyer. "Governance in Decentralised Autonomous Organisations Related". In: (), pp. 3–9.

[16] Mateusz Mikuła and Mariusz Dzieńkowski. "Comparison of REST and GraphQL web technology performance". In: *Journal of Computer Sciences Institute* 16.August (2020), pp. 309–316. DOI: 10.35784/jcsi.2077.

[17] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: *Transforming Government: People, Process and Policy* 0.0 (2008), p. 12. URL: https://bitcoin.org/bitcoin.pdf.

[18] Mridul Nandi. "Designs of Efficient Secure Large Hash Values." In: *IACR Cryptology ePrint Archive* 2004 (2004), p. 296. URL: http://dblp.uni-trier.de/db/journals/iacr/iacr2004.html#Nandi04.

[19] Michael Nofer et al. "Blockchain". In: *Business and Information Systems Engineering* 59.3 (2017), pp. 183–187. ISSN: 18670202. DOI: 10.1007/s12599-017-0467-3.

[20] Ken Peffers et al. "A design science research methodology for information systems research". In: *Journal of Management Information Systems* 24.3 (2007), pp. 45–77. ISSN: 07421222. DOI: 10.2753/MIS0742-1222240302.

[21] Matheus Seabra, Marcos Felipe Nazário, and Gustavo Pinto. "REST or GraphQL?" In: (2019), pp. 123–132. DOI: 10.1145/3357141.3357149.

[22] Evrim Tan, Stanislav Mahula, and Joep Crompvoets. "Blockchain governance in the public sector: A conceptual framework for public management". In: *Government Information Quarterly* 39.1 (2021), p. 101625. ISSN: 0740624X. DOI: 10.1016/j.giq.2021.101625. URL: https://doi.org/10.1016/j.giq.2021.101625.

[23] Gavin Wood. "Ethereum: a secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151 (2014), pp. 1–32.

[24] Dylan Yaga et al. "Blockchain Technology Overview". In: (2019). DOI: 10.6028/NIST.IR.8202. arXiv: 1906.11078. URL: http://arxiv.org/abs/1906.11078%0Ahttp://dx.doi.org/10.6028/NIST.IR.8202.

# Appendix A

# networkRequest

```
1  package networkRequest
2
3  import "math/big"
4
5  type Request struct {
6    NetworkID         string
7    AssetID           string
8    ProposalID        big.Int
9    Vote              big.Int
10   UserID            int64
11   Type              int
12   Quantity          int64
13   TransferReceiver  string
14   Data              string
15   Address           string
16   PrivateKey        string
17 }
```

Listing A.1: Requests.go

# Appendix B

# Compound Smart Contract 1 (Comp.sol)

```
1     function delegate(address delegatee) public {
2         return _delegate(msg.sender, delegatee);
3     }
4
5     function delegateBySig(address delegatee, uint nonce, uint expiry,
    uint8 v, bytes32 r, bytes32 s) public {
6         bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH,
    keccak256(bytes(name)), getChainId(), address(this)));
7         bytes32 structHash = keccak256(abi.encode(DELEGATION_TYPEHASH,
    delegatee, nonce, expiry));
8         bytes32 digest = keccak256(abi.encodePacked("\x19\x01",
    domainSeparator, structHash));
9         address signatory = ecrecover(digest, v, r, s);
10        require(signatory != address(0), "Comp::delegateBySig: invalid
    signature");
11        require(nonce == nonces[signatory]++, "Comp::delegateBySig:
    invalid nonce");
12        require(now <= expiry, "Comp::delegateBySig: signature expired")
    ;
13        return _delegate(signatory, delegatee);
14    }
15
16    function _delegate(address delegator, address delegatee) internal {
17        address currentDelegate = delegates[delegator];
18        uint96 delegatorBalance = balances[delegator];
19        delegates[delegator] = delegatee;
20
21        emit DelegateChanged(delegator, currentDelegate, delegatee);
22
23        _moveDelegates(currentDelegate, delegatee, delegatorBalance);
24    }
```

Listing B.1: Compound Smart Contract 1 (Comp.sol)

# Appendix C

# Compound Smart Contract 2 (GovernorAlpha.sol)

```
    function castVote(uint proposalId, bool support) public {
        return _castVote(msg.sender, proposalId, support);
    }

    function castVoteBySig(uint proposalId, bool support, uint8 v,
    bytes32 r, bytes32 s) public {
        bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH,
    keccak256(bytes(name)), getChainId(), address(this)));
        bytes32 structHash = keccak256(abi.encode(BALLOT_TYPEHASH,
    proposalId, support));
        bytes32 digest = keccak256(abi.encodePacked("\x19\x01",
    domainSeparator, structHash));
        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "GovernorAlpha::castVoteBySig:
    invalid signature");
        return _castVote(signatory, proposalId, support);
    }

    function _castVote(address voter, uint proposalId, bool support)
    internal {
        require(state(proposalId) == ProposalState.Active, "
    GovernorAlpha::_castVote: voting is closed");
        Proposal storage proposal = proposals[proposalId];
        Receipt storage receipt = proposal.receipts[voter];
        require(receipt.hasVoted == false, "GovernorAlpha::_castVote:
    voter already voted");
        uint96 votes = comp.getPriorVotes(voter, proposal.startBlock);

        if (support) {
            proposal.forVotes = add256(proposal.forVotes, votes);
        } else {
            proposal.againstVotes = add256(proposal.againstVotes, votes)
    ;
        }

        receipt.hasVoted = true;
        receipt.support = support;
        receipt.votes = votes;

        emit VoteCast(voter, proposalId, support, votes);
    }

```

```
34      function propose(address[] memory targets, uint[] memory values,
        string[] memory signatures, bytes[] memory calldatas, string memory
        description) public returns (uint) {
35          require(comp.getPriorVotes(msg.sender, sub256(block.number, 1))
        > proposalThreshold(), "GovernorAlpha::propose: proposer votes below
        proposal threshold");
36          require(targets.length == values.length && targets.length ==
        signatures.length && targets.length == calldatas.length, "
        GovernorAlpha::propose: proposal function information arity mismatch"
        );
37          require(targets.length != 0, "GovernorAlpha::propose: must
        provide actions");
38          require(targets.length <= proposalMaxOperations(), "
        GovernorAlpha::propose: too many actions");
39
40          uint latestProposalId = latestProposalIds[msg.sender];
41          if (latestProposalId != 0) {
42            ProposalState proposersLatestProposalState = state(
        latestProposalId);
43            require(proposersLatestProposalState != ProposalState.Active,
        "GovernorAlpha::propose: one live proposal per proposer, found an
        already active proposal");
44            require(proposersLatestProposalState != ProposalState.Pending,
         "GovernorAlpha::propose: one live proposal per proposer, found an
        already pending proposal");
45          }
46
47          uint startBlock = add256(block.number, votingDelay());
48          uint endBlock = add256(startBlock, votingPeriod());
49
50          proposalCount++;
51          uint proposalId = proposalCount;
52          Proposal storage newProposal = proposals[proposalId];
53          // This should never happen but add a check in case.
54          require(newProposal.id == 0, "GovernorAlpha::propose: ProposalID
         collsion");
55          newProposal.id = proposalId;
56          newProposal.proposer = msg.sender;
57          newProposal.eta = 0;
58          newProposal.targets = targets;
59          newProposal.values = values;
60          newProposal.signatures = signatures;
61          newProposal.calldatas = calldatas;
62          newProposal.startBlock = startBlock;
63          newProposal.endBlock = endBlock;
64          newProposal.forVotes = 0;
65          newProposal.againstVotes = 0;
66          newProposal.canceled = false;
67          newProposal.executed = false;
68
69          latestProposalIds[newProposal.proposer] = newProposal.id;
70
71          emit ProposalCreated(newProposal.id, msg.sender, targets, values
        , signatures, calldatas, startBlock, endBlock, description);
72          return newProposal.id;
73      }
```

Listing C.1: Compound Smart Contract 2 (GovernorAlpha.sol)

# Appendix D

# MakerDAO Smart Contract 1 (token.sol)

```
1     function approve(address guy) external returns (bool) {
2         return approve(guy, uint(-1));
3     }
4
5     function approve(address guy, uint wad) public stoppable returns (
      bool) {
6         allowance[msg.sender][guy] = wad;
7
8         emit Approval(msg.sender, guy, wad);
9
10        return true;
11    }
```

Listing D.1: MakerDAO Smart Contract 1 (token.sol)

# Appendix E

# MakerDAO Smart Contract 2 (chief.sol)

```
    function lock(uint wad)
        public
        note
    {
        last[msg.sender] = block.number;
        GOV.pull(msg.sender, wad);
        IOU.mint(msg.sender, wad);
        deposits[msg.sender] = add(deposits[msg.sender], wad);
        addWeight(wad, votes[msg.sender]);
    }

    function free(uint wad)
        public
        note
    {
        require(block.number > last[msg.sender]);
        deposits[msg.sender] = sub(deposits[msg.sender], wad);
        subWeight(wad, votes[msg.sender]);
        IOU.burn(msg.sender, wad);
        GOV.push(msg.sender, wad);
    }

    function vote(address[] memory yays) public returns (bytes32)
        // note  both sub-calls note
    {
        bytes32 slate = etch(yays);
        vote(slate);
        return slate;
    }

    function etch(address[] memory yays)
        public
        note
        returns (bytes32 slate)
    {
        require( yays.length <= MAX_YAYS );
        requireByteOrderedSet(yays);

        bytes32 hash = keccak256(abi.encodePacked(yays));
        slates[hash] = yays;
        emit Etch(hash);
        return hash;
    }

```

```
45      function vote(bytes32 slate)
46          public
47          note
48      {
49          require(slates[slate].length > 0 ||
50              slate == 0
      xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470, "
      ds-chief-invalid-slate");
51          uint weight = deposits[msg.sender];
52          subWeight(weight, votes[msg.sender]);
53          votes[msg.sender] = slate;
54          addWeight(weight, votes[msg.sender]);
55      }
```

Listing E.1: MakerDAO Smart Contract 2 (chief.sol)

# Appendix F

# Aave Smart Contract 1 (GovernancePowerDelegationERC20.sol)

```
1    function delegate(address delegatee) external override {
2      _delegateByType(msg.sender, delegatee, DelegationType.VOTING_POWER);
3      _delegateByType(msg.sender, delegatee, DelegationType.
       PROPOSITION_POWER);
4    }
5
6    function delegateByType(address delegatee, DelegationType
     delegationType) external override {
7      _delegateByType(msg.sender, delegatee, delegationType);
8    }
9
10   function _delegateByType(
11     address delegator,
12     address delegatee,
13     DelegationType delegationType
14   ) internal {
15     require(delegatee != address(0), 'INVALID_DELEGATEE');
16
17     (, , mapping(address => address) storage delegates) =
       _getDelegationDataByType(delegationType);
18
19     uint256 delegatorBalance = balanceOf(delegator);
20
21     address previousDelegatee = _getDelegatee(delegator, delegates);
22
23     delegates[delegator] = delegatee;
24
25     _moveDelegatesByType(previousDelegatee, delegatee, delegatorBalance,
       delegationType);
26     emit DelegateChanged(delegator, delegatee, delegationType);
27   }
```

LISTING F.1: Aave Smart Contract 1 (GovernancePowerDelegationERC20.sol)

# Appendix G

# Aave Smart Contract 2 (GovernancePowerDelegation-ERC20.sol)

```
1    function submitVote(uint256 proposalId, bool support) external
     override {
2      return _submitVote(msg.sender, proposalId, support);
3    }
4
5    function submitVoteBySignature(
6      uint256 proposalId,
7      bool support,
8      uint8 v,
9      bytes32 r,
10     bytes32 s
11   ) external override {
12     bytes32 digest = keccak256(
13       abi.encodePacked(
14         '\x19\x01',
15         keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(NAME)),
     getChainId(), address(this))),
16         keccak256(abi.encode(VOTE_EMITTED_TYPEHASH, proposalId, support)
     )
17       )
18     );
19     address signer = ecrecover(digest, v, r, s);
20     require(signer != address(0), 'INVALID_SIGNATURE');
21     return _submitVote(signer, proposalId, support);
22   }
23
24   function _submitVote(
25     address voter,
26     uint256 proposalId,
27     bool support
28   ) internal {
29     require(getProposalState(proposalId) == ProposalState.Active, '
     VOTING_CLOSED');
30     Proposal storage proposal = _proposals[proposalId];
31     Vote storage vote = proposal.votes[voter];
32
33     require(vote.votingPower == 0, 'VOTE_ALREADY_SUBMITTED');
34
35     uint256 votingPower = IVotingStrategy(proposal.strategy).
     getVotingPowerAt(
36       voter,
```

```
37        proposal.startBlock
38      );
39
40      if (support) {
41        proposal.forVotes = proposal.forVotes.add(votingPower);
42      } else {
43        proposal.againstVotes = proposal.againstVotes.add(votingPower);
44      }
45
46      vote.support = support;
47      vote.votingPower = uint248(votingPower);
48
49      emit VoteEmitted(proposalId, voter, support, votingPower);
50    }
```

LISTING G.1: Aave Smart Contract 2 (GovernancePowerDelegationERC20.sol)

# Appendix H

# DYDX Smart Contract 1 (GovernancePowerDelegation-ERC20Mixin.sol)

```
    function delegate(address delegatee) external override {
    _delegateByType(msg.sender, delegatee, DelegationType.VOTING_POWER);
    _delegateByType(msg.sender, delegatee, DelegationType.
    PROPOSITION_POWER);
  }

   function delegateByType(address delegatee, DelegationType
    delegationType) external override {
    _delegateByType(msg.sender, delegatee, delegationType);
  }

  function _delegateByType(
    address delegator,
    address delegatee,
    DelegationType delegationType
  ) internal {
    require(delegatee != address(0), 'INVALID_DELEGATEE');

    (, , mapping(address => address) storage delegates) =
    _getDelegationDataByType(delegationType);

    uint256 delegatorBalance = balanceOf(delegator);

    address previousDelegatee = _getDelegatee(delegator, delegates);

    delegates[delegator] = delegatee;

    _moveDelegatesByType(previousDelegatee, delegatee, delegatorBalance,
     delegationType);
    emit DelegateChanged(delegator, delegatee, delegationType);
  }
```

LISTING H.1: DYDX Smart Contract 1 (GovernancePowerDelegationERC20Mixin.sol)

# Appendix I

# DYDX Smart Contract 2 (GovernancePowerDelegation-ERC20Mixin.sol)

```
    function submitVote(uint256 proposalId, bool support) external override{
    return _submitVote(msg.sender, proposalId, support);
  }

  function submitVoteBySignature(uint256 proposalId, bool support, uint8 v, bytes32 r, bytes32 s) external override {
    bytes32 digest = keccak256(
      abi.encodePacked(
        '\x19\x01',
        keccak256(
          abi.encode(
            DOMAIN_TYPEHASH,
            keccak256(bytes(EIP712_DOMAIN_NAME)),
            getChainId(),
            address(this)
          )
        ),
        keccak256(abi.encode(VOTE_EMITTED_TYPEHASH, proposalId, support))
      )
    );
    address signer = ecrecover(digest, v, r, s);
    require(
      signer != address(0),
      'INVALID_SIGNATURE'
    );
    return _submitVote(signer, proposalId, support);
  }

  function _submitVote(address voter, uint256 proposalId, bool support) internal{
    require(
      getProposalState(proposalId) == ProposalState.Active,
      'VOTING_CLOSED'
    );
    Proposal storage proposal = _proposals[proposalId];
    Vote storage vote = proposal.votes[voter];

    require(
      vote.votingPower == 0,
```

```
38        'VOTE_ALREADY_SUBMITTED'
39      );
40
41      uint256 votingPower = IVotingStrategy(proposal.strategy).
        getVotingPowerAt(
42        voter,
43        proposal.startBlock
44      );
45
46      if (support) {
47        proposal.forVotes = proposal.forVotes.add(votingPower);
48      } else {
49        proposal.againstVotes = proposal.againstVotes.add(votingPower);
50      }
51
52      vote.support = support;
53      vote.votingPower = uint248(votingPower);
54
55      emit VoteEmitted(proposalId, voter, support, votingPower);
56    }
```

Listing I.1: DYDX Smart Contract 2 (GovernancePowerDelegationERC20Mixin.sol)

# Appendix J

# Governance Proposal Interfaces and Identifiers

```
pragma solidity ^0.8.0;

interface GovernanceStandardCore {

    event ProposalCreated(uint256 _id, address _proposer, address[]
    _targets, uint256[] _values, string[] _signatures, bytes[] _calldatas
    );

    event VoteCast(address _voter, uint _proposeId, uint8 _support,
    string _data);

    event VoteDelegated(address _delegator, address _delegatee, uint256
    _value);

    function propose(
    address[] memory _targets,
    uint256[] memory _values,
    string[] memory _signatures,
    bytes[] memory _calldatas
    ) external returns (uint256);

    function vote(
    uint _proposalId,
    uint8 _support,
    string memory _data
    ) external;

    function delegate(
    address _delegatee,
    uint256 _value
    ) external;
}

interface GovernanceStandardLocking {

    event Locked(address _user, uint256 _value);

    event Unlocked(address _user, uint256 _value);

    function lock(uint256 _value) external;

    function unlock(uint256 _value) external;
}

```

```
41 interface GovernanceStandardActionBySignature {
42
43     function proposeBySig(
44     address[] memory _targets,
45     uint256[] memory _values,
46     string[] memory _signatures,
47     bytes[] memory _calldatas,
48     uint _nonce,
49     uint _expiry,
50     uint8 _v,
51     bytes32 _r,
52     bytes32 _s
53     ) external returns (uint256);
54
55     function voteBySig(
56     uint _proposalId,
57     uint8 _support,
58     string memory _data,
59     uint _nonce,
60     uint _expiry,
61     uint8 _v,
62     bytes32 _r,
63     bytes32 _s
64     ) external;
65
66     function delegateBySig(
67     address _delegatee,
68     uint256 _value,
69     uint _nonce,
70     uint _expiry,
71     uint8 _v,
72     bytes32 _r,
73     bytes32 _s
74     ) external;
75
76 }
77
78 interface ERC165 {
79
80     function supportsInterface(bytes4 interfaceID) external view returns
       (bool);
81 }
82
83
84 contract Selector {
85     function calculateGovernanceStandardCoreSelector() public pure
      returns (bytes4) {
86         GovernanceStandardCore i;
87         return i.propose.selector ^ i.vote.selector ^ i.delegate.
      selector;
88     }
89
90     function calculateERC165eSelector() public pure returns (bytes4) {
91         ERC165 i;
92         return i.supportsInterface.selector;
93     }
94
95     function calculateGovernanceStandardLockingSelector() public pure
      returns (bytes4) {
96         GovernanceStandardLocking i;
97         return i.lock.selector ^ i.unlock.selector;
```

```
98      }
99
100     function calculateGovernanceStandardActionBySignatureSelector()
        public pure returns (bytes4) {
101         GovernanceStandardActionBySignature i;
102         return i.proposeBySig.selector ^ i.voteBySig.selector ^ i.
        delegateBySig.selector;
103     }
104 }
```

Listing J.1: Governance Proposal Interfaces and Identifiers

# Appendix K

# Governance Standard Proposal Example

```
pragma solidity ^0.8.0;

import {GovernanceStandardCore} from './GovernanceInterface.sol';
import {ERC165} from './ERC165.sol';

contract GovernanceLogicContract is GovernanceStandardCore, ERC165  {

    uint256 private proposalsCount;
    mapping(uint256 => Proposal) public proposals;
    mapping(address => address) public delegatees;
    uint256 public proposalCount;

    struct Proposal {
    uint256 id;
    address[] targets;
    uint256[] values;
    string[] signatures;
    bytes[] calldatas;
    uint256 yays;
    uint256 nays;
  }


    function propose(address[] memory _targets, uint256[] memory _values
    , string[] memory _signatures, bytes[] memory _calldatas) public
    override returns (uint256){
        uint256 newProposalCount = proposalCount + 1;
        proposalCount++;
        Proposal storage newProposal = proposals[newProposalCount];
        newProposal.id = newProposalCount;
        newProposal.targets = _targets;
        newProposal.values = _values;
        newProposal.signatures = _signatures;
        newProposal.calldatas = _calldatas;
        newProposal.yays = 0;
        newProposal.nays = 0;

        emit ProposalCreated(
            newProposalCount,
            msg.sender,
            _targets,
            _values,
            _signatures,
            _calldatas
```

```
43          );
44      }
45
46      function vote(uint _proposalId, uint8 _support, string memory _data)
         public override{
47          uint256 currentNays = proposals[_proposalId].nays;
48          uint256 currentYays = proposals[_proposalId].yays;
49          if(_support == 1) {
50              proposals[_proposalId].yays = currentYays + 1;
51          }
52          else if(_support == 2){
53             proposals[_proposalId].nays = currentNays + 1;
54          }
55
56          emit VoteCast(
57              msg.sender,
58              _proposalId,
59              _support,
60              _data
61          );
62      }
63
64      function delegate(address _delegatee, uint256 _value) public
        override{
65          delegatees[msg.sender] = _delegatee;
66
67          emit VoteDelegated(
68              msg.sender,
69              _delegatee,
70              _value
71          );
72      }
73
74      function supportsInterface(bytes4 interfaceID) public override view
        returns (bool){
75          if(interfaceID == 0xdf1132f0 || interfaceID == 0x01ffc9a7){
76              return true;
77          }else{
78              return false;
79          }
80
81
82      }
83  }
```

Listing K.1: Governance Standard Proposal Example

# Appendix L

# Governance Standard Proposal Proxy Pattern Interface Example

```
pragma solidity ^0.8.0;

import {GovernanceStandardCore} from './GovernanceInterface.sol';
import {ERC165} from './ERC165.sol';

contract GovernanceContract is GovernanceStandardCore, ERC165{

    address private _governanceLogicAddress;
    mapping(uint256 => Proposal) public proposals;
    mapping(address => address) public delegatees;
    uint256 public proposalCount;

    struct Proposal {
    uint256 id;
    address[] targets;
    uint256[] values;
    string[] signatures;
    bytes[] calldatas;
    uint256 yays;
    uint256 nays;
    }

    constructor(address governanceLogicAddress_) {

        _governanceLogicAddress = governanceLogicAddress_;

    }

    function updateGovernanceLogic(address newGovernanceAddress_) public {
        _governanceLogicAddress = newGovernanceAddress_;
    }

    function propose(address[] memory _targets, uint256[] memory _values, string[] memory _signatures, bytes[] memory _calldatas) public override returns (uint256){

        (bool success, bytes memory result) = _governanceLogicAddress.delegatecall(abi.encodeWithSignature("propose(address[],uint256[],string[],bytes[])",_targets, _values, _signatures, _calldatas));

        require(success == true, "Unknown error delegating call to logic contract.");

```

```
39          emit ProposalCreated(
40              proposalCount,
41              msg.sender,
42              _targets,
43              _values,
44              _signatures,
45              _calldatas
46          );
47
48      }
49
50      function vote(uint _proposalId, uint8 _support, string memory _data)
         public override{
51
52          (bool success, bytes memory result) = _governanceLogicAddress.
        delegatecall(abi.encodeWithSignature("castVote(uint256,uint8,string)"
        ,_proposalId,_support,_data));
53
54          require(success == true, "Unknown error delegating call to logic
         contract.");
55
56          emit VoteCast(
57              msg.sender,
58              _proposalId,
59              _support,
60              _data
61          );
62
63      }
64
65      function delegate(address _delegatee, uint256 _value) public
        override{
66
67          (bool success, bytes memory result) = _governanceLogicAddress.
        delegatecall(abi.encodeWithSignature("delegate(address,uint256)",
        _delegatee,_value));
68
69          require(success == true, "Unknown error delegating call to logic
         contract.");
70
71          emit VoteDelegated(
72              msg.sender,
73              _delegatee,
74              _value
75          );
76
77      }
78
79      function supportsInterface(bytes4 interfaceID) public override view
        returns (bool){
80          if(interfaceID == 0xdf1132f0 || interfaceID == 0x01ffc9a7){
81              return true;
82          }else{
83              return false;
84          }
85
86
87      }
88
89  }
```

LISTING L.1: Governance Standard Proposal Proxy Pattern Interface Example

# Appendix M

# Governance Standard Proposal Proxy Pattern Logic Example

```
pragma solidity ^0.8.0;

contract GovernanceLogicContract  {

    address private _mocGovernanceAddress = 0
    x0000000000000000000000000000000000000000;
    mapping(uint256 => Proposal) public proposals;
    mapping(address => address) public delegatees;
    uint256 public proposalCount;

    struct Proposal {
    uint256 id;
    address[] targets;
    uint256[] values;
    string[] signatures;
    bytes[] calldatas;
    uint256 yays;
    uint256 nays;
    }

    event ProposalCreated(uint256 _id, address _proposer, address[]
    _targets, uint256[] _values, string[] _signatures, bytes[] _calldatas
    );

    event VoteCast(address _voter, uint _proposeId, uint8 _support,
    string _data);

    event VoteDelegated(address _delegator, address _delegatee, uint256
    _value);

    function propose(address[] memory _targets, uint256[] memory _values
    , string[] memory _signatures, bytes[] memory _calldatas) external {
        uint256 newProposalCount = proposalCount + 1;
        proposalCount++;
        Proposal storage newProposal = proposals[proposalCount];
        newProposal.id = newProposalCount;
        newProposal.targets = _targets;
        newProposal.values = _values;
        newProposal.signatures = _signatures;
        newProposal.calldatas = _calldatas;
        newProposal.yays = 0;
        newProposal.nays = 0;
    }

```

```
39      function castVote(uint proposalId, uint8 support, string memory data
        ) external {
40          Proposal storage proposal = proposals[1];
41          if(support == 1) {
42              proposal.yays += 1;
43          }
44          else if(support == 2){
45             proposal.nays += 1;
46          }
47      }
48
49      function delegate(address delegatee, uint256 value) external {
50          delegatees[msg.sender] = delegatee;
51      }
52  }
```

Listing M.1: Governance Standard Proposal Proxy Pattern Logic Example

# Appendix N

# Service implementation of governance standard

```
package governanceLogic

import (
  "errors"
  "strconv"

  Request "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/networkRequest"
  Signer "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/offlinesigner"
  ABI "github.com/hchuva/ExtensibleGovernance/CollapsedService/pkg/abiEncoder"
)

type GovernanceStandard struct {
  AssetName       string
  AssetID         string
  AssetNetwork    string
  VoteAddress     string
  DelegateAddress string
  Signer          Signer.Service
}

func (action GovernanceStandard) Vote(_request Request.Request) ([]AbstractedSmartContractAction, error) {

  var returnArray []AbstractedSmartContractAction

  proposalParameter := ABI.FunctionParameter{Value: strconv.FormatInt(_request.ProposalID.Int64(), 10), ValueType: "Int"}
  supportParameter := ABI.FunctionParameter{Value: strconv.FormatInt(_request.Vote.Int64(), 10), ValueType: "Int"}
  dataParameter := ABI.FunctionParameter{Value: "0x0", ValueType: "String"}

  data := ABI.EncodeSig("submitVote(uint256,uint8,string)", proposalParameter, supportParameter, dataParameter)

  returnArray = append(returnArray, AbstractedSmartContractAction{Data: data, Address: action.VoteAddress})

  return returnArray, nil
}

```

```
36 func (action GovernanceStandard) Delegate(_request Request.Request) ([]
       AbstractedSmartContractAction, error) {
37
38   var returnArray []AbstractedSmartContractAction
39
40   delegateeParameter := ABI.FunctionParameter{Value: _request.
       TransferReceiver, ValueType: "Address"}
41   valueParameter := ABI.FunctionParameter{Value: strconv.FormatInt(
       _request.Quantity, 10), ValueType: "Int"}
42
43   data := ABI.EncodeSig("delegate(address,uint256)", delegateeParameter,
        valueParameter)
44
45   returnArray = append(returnArray, AbstractedSmartContractAction{Data:
       data, Address: action.DelegateAddress})
46
47   return returnArray, nil
48 }
49
50 func (action GovernanceStandard) Lock(_request Request.Request) ([]
       AbstractedSmartContractAction, error) {
51   return nil, errors.New("assetHandler: Asset named " + _request.AssetID
        + " does not contain Lock function")
52 }
53
54 func (action GovernanceStandard) Unlock(_request Request.Request) ([]
       AbstractedSmartContractAction, error) {
55   return nil, errors.New("assetHandler: Asset named " + _request.AssetID
        + " does not contain Unlock function")
56 }
57
58 func NewGovernanceStandard(signer Signer.Service)
       AbstractedSmartContract {
59   return AssetTemplate{
60     AssetName:       "GovernanceStandard",
61     AssetID:         "GSD",
62     AssetNetwork:    "ETH",
63     VoteAddress:     "0x0000000000000000000000000000000000000000",
64     DelegateAddress: "0x0000000000000000000000000000000000000000",
65     Signer:          signer,
66   }
67 }
```

Listing N.1: Service implementation of governance standard

# Appendix O

# Service usage of governance standard

```
1  package governanceLogic
2
3  import Signer "github.com/hchuva/ExtensibleGovernance/CollapsedService/
       internal/offlinesigner"
4
5  func NewGovernanceExample(signer Signer.Service) AbstractedSmartContract
       {
6    return GovernanceStandard{
7      AssetName:       "StandardExample",
8      AssetID:         "SEE",
9      AssetNetwork:    "ETH",
10     VoteAddress:     "0xb9a732a1ADaC81f463f40CF013Ac9738465Aa614",
11     DelegateAddress: "0x491Ee16230aF6876fD67eFBf16229Fb2672E35aF",
12     Signer:          signer,
13   }
14 }
```

Listing O.1: Service usage of governance standard

# Appendix P

# Compound Project Implementation configuration file

```
package governanceLogic

import (
  "errors"
  "strconv"

  Request "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/networkRequest"
  Signer "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/offlinesigner"
  ABI "github.com/hchuva/ExtensibleGovernance/CollapsedService/pkg/abiEncoder"
)

type Compound struct {
  AssetName       string
  AssetID         string
  AssetNetwork    string
  VoteAddress     string
  DelegateAddress string
  Signer          Signer.Service
}

func (action Compound) Vote(_request Request.Request) ([]AbstractedSmartContractAction, error) {

  var returnArray []AbstractedSmartContractAction

  _request.TransferReceiver = action.Signer.GetAddress(_request.PrivateKey)

  returnArray = append(returnArray, AbstractedSmartContractAction{Data: action.delegate(_request), Address: action.DelegateAddress})

  proposalParameter := ABI.FunctionParameter{Value: strconv.FormatInt(_request.ProposalID.Int64(), 10), ValueType: "Int"}
  supportParameter := ABI.FunctionParameter{Value: strconv.FormatInt(_request.Vote.Int64(), 10), ValueType: "Int"}

  data := ABI.EncodeSig("castVote(uint256,bool)", proposalParameter, supportParameter)

  returnArray = append(returnArray, AbstractedSmartContractAction{Data: data, Address: action.VoteAddress})
```

```
35
36    return returnArray, nil
37  }
38
39  func (action Compound) Delegate(_request Request.Request) ([]
      AbstractedSmartContractAction, error) {
40
41    var returnArray []AbstractedSmartContractAction
42
43    returnArray = append(returnArray, AbstractedSmartContractAction{Data:
        action.delegate(_request), Address: action.DelegateAddress})
44
45    return returnArray, nil
46  }
47
48  func (action Compound) delegate(_request Request.Request) []byte {
49
50    delegateeParameter := ABI.FunctionParameter{Value: _request.
        TransferReceiver, ValueType: "Address"}
51
52    data := ABI.EncodeSig("delegate(address)", delegateeParameter)
53
54    return data
55  }
56
57  func (action Compound) Lock(_request Request.Request) ([]
      AbstractedSmartContractAction, error) {
58
59    return nil, errors.New("assetHandler: Asset named " + _request.AssetID
        + " does not contain Lock function")
60  }
61
62  func (action Compound) Unlock(_request Request.Request) ([]
      AbstractedSmartContractAction, error) {
63
64    return nil, errors.New("assetHandler: Asset named " + _request.AssetID
        + " does not contain Unlock function")
65  }
66
67  func NewCompound(signer Signer.Service) AbstractedSmartContract {
68    return Compound{
69      AssetName:       "Compound",
70      AssetID:         "CMP",
71      AssetNetwork:    "ETH",
72      VoteAddress:     "0xb9a732a1ADaC81f463f40CF013Ac9738465Aa614",
73      DelegateAddress: "0x491Ee16230aF6876fD67eFBf16229Fb2672E35aF",
74      Signer:          signer,
75    }
76  }
```

Listing P.1: Compound Project Implementation configuration file

# Appendix Q

# Aave Project Implementation configuration file

```
package governanceLogic

import (
  "errors"
  "strconv"

  Request "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/networkRequest"
  Signer "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/offlinesigner"
  ABI "github.com/hchuva/ExtensibleGovernance/CollapsedService/pkg/abiEncoder"
)

type Aave struct {
  AssetName       string
  AssetID         string
  AssetNetwork    string
  VoteAddress     string
  DelegateAddress string
  Signer          Signer.Service
}

func (action Aave) Vote(_request Request.Request) ([]AbstractedSmartContractAction, error) {

  var returnArray []AbstractedSmartContractAction

  proposalParameter := ABI.FunctionParameter{Value: strconv.FormatInt(_request.ProposalID.Int64(), 10), ValueType: "Int"}
  supportParameter := ABI.FunctionParameter{Value: strconv.FormatInt(_request.Vote.Int64(), 10), ValueType: "Int"}

  data := ABI.EncodeSig("submitVote(uint256,bool)", proposalParameter, supportParameter)

  returnArray = append(returnArray, AbstractedSmartContractAction{Data: data, Address: action.VoteAddress})

  return returnArray, nil
}

func (action Aave) Delegate(_request Request.Request) ([]AbstractedSmartContractAction, error) {
```

```
36
37    var returnArray []AbstractedSmartContractAction
38
39    delegateeParameter := ABI.FunctionParameter{Value: _request.
        TransferReceiver, ValueType: "Address"}
40
41    data := ABI.EncodeSig("delegate(address)", delegateeParameter)
42
43    returnArray = append(returnArray, AbstractedSmartContractAction{Data:
        data, Address: action.DelegateAddress})
44
45    return returnArray, nil
46  }
47
48  func (action Aave) Lock(_request Request.Request) ([]
      AbstractedSmartContractAction, error) {
49
50    return nil, errors.New("assetHandler: Asset named " + _request.AssetID
        + " does not contain Lock function")
51  }
52
53  func (action Aave) Unlock(_request Request.Request) ([]
      AbstractedSmartContractAction, error) {
54
55    return nil, errors.New("assetHandler: Asset named " + _request.AssetID
        + " does not contain Unlock function")
56  }
57
58  func NewAave(signer Signer.Service) AbstractedSmartContract {
59    return Aave{
60      AssetName:       "Aave",
61      AssetID:         "AAVE",
62      AssetNetwork:    "ETH",
63      VoteAddress:     "0xb9a732a1ADaC81f463f40CF013Ac9738465Aa614",
64      DelegateAddress: "0x491Ee16230aF6876fD67eFBf16229Fb2672E35aF",
65      Signer:          signer,
66    }
67  }
```

Listing Q.1: Aave Project Implementation configuration file

# Appendix R

# MakerDAO Project Implementation configuration file

```
package governanceLogic

import (
  "strconv"

  Request "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/networkRequest"
  Signer "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/offlinesigner"
  ABI "github.com/hchuva/ExtensibleGovernance/CollapsedService/pkg/abiEncoder"
)

type Maker struct {
  AssetName    string
  AssetID      string
  AssetNetwork string
  VoteAddress  string
  MKRAddress   string
  IOUAddress   string
  Signer       Signer.Service
}

func (action Maker) Vote(_request Request.Request) ([]AbstractedSmartContractAction, error) {

  var returnArray []AbstractedSmartContractAction

  slateParameter := ABI.FunctionParameter{Value: _request.TransferReceiver, ValueType: "Byte32"}

  data := ABI.EncodeSig("vote(bytes32)", slateParameter)

  returnArray = append(returnArray, AbstractedSmartContractAction{Data: data, Address: action.VoteAddress})

  return returnArray, nil
}

func (action Maker) Delegate(_request Request.Request) ([]AbstractedSmartContractAction, error) {

  var returnArray []AbstractedSmartContractAction

```

```
38    returnArray = append(returnArray, AbstractedSmartContractAction{Data:
        action.delegate(_request), Address: action.MKRAddress})
39
40    return returnArray, nil
41  }
42
43  func (action Maker) delegate(_request Request.Request) []byte {
44
45    delegateeParameter := ABI.FunctionParameter{Value: _request.
        TransferReceiver, ValueType: "Address"}
46
47    data := ABI.EncodeSig("approve(address)", delegateeParameter)
48
49    return data
50  }
51
52  func (action Maker) Lock(_request Request.Request) ([]
      AbstractedSmartContractAction, error) {
53
54    var returnArray []AbstractedSmartContractAction
55
56    _request.TransferReceiver = action.VoteAddress
57
58    returnArray = append(returnArray, AbstractedSmartContractAction{Data:
        action.delegate(_request), Address: action.MKRAddress})
59
60    wadParameter := ABI.FunctionParameter{Value: strconv.FormatInt(
        _request.Quantity, 10), ValueType: "Int"}
61
62    data := ABI.EncodeSig("lock(uint256)", wadParameter)
63
64    returnArray = append(returnArray, AbstractedSmartContractAction{Data:
        data, Address: action.VoteAddress})
65
66    return returnArray, nil
67  }
68
69  func (action Maker) Unlock(_request Request.Request) ([]
      AbstractedSmartContractAction, error) {
70
71    var returnArray []AbstractedSmartContractAction
72
73    _request.TransferReceiver = action.VoteAddress
74
75    returnArray = append(returnArray, AbstractedSmartContractAction{Data:
        action.delegate(_request), Address: action.IOUAddress})
76
77    wadParameter := ABI.FunctionParameter{Value: strconv.FormatInt(
        _request.Quantity, 10), ValueType: "Int"}
78
79    data := ABI.EncodeSig("free(uint256)", wadParameter)
80
81    returnArray = append(returnArray, AbstractedSmartContractAction{Data:
        data, Address: action.VoteAddress})
82
83    return returnArray, nil
84  }
85
86  func NewMaker(signer Signer.Service) AbstractedSmartContract {
87    return Maker{
88      AssetName:    "Maker",
```

```
    AssetID:       "MKR",
    AssetNetwork: "ETH",
    VoteAddress:   "0xb9a732a1ADaC81f463f40CF013Ac9738465Aa614",
    MKRAddress:    "0x491Ee16230aF6876fD67eFBf16229Fb2672E35aF",
    IOUAddress:    "0x491Ee16230aF6876fD67eFBf16229Fb2672E35aF",
    Signer:        signer,
  }
}
```

Listing R.1: MakerDAO Project Implementation configuration file

# Appendix S

# DYDX Project Implementation configuration file

```
package governanceLogic

import (
  "encoding/hex"
  "errors"
  "math/big"
  "strconv"

  "github.com/ethereum/go-ethereum/crypto"
  Request "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/networkRequest"
  Signer "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/offlinesigner"
  ABI "github.com/hchuva/ExtensibleGovernance/CollapsedService/pkg/abiEncoder"
)

type DYDX struct {
  AssetName       string
  AssetID         string
  AssetNetwork    string
  VoteAddress     string
  DelegateAddress string
  Signer          Signer.Service
}

func (action DYDX) Vote(_request Request.Request) ([]AbstractedSmartContractAction, error) {

  var returnArray []AbstractedSmartContractAction

  proposalParameter := ABI.FunctionParameter{Value: strconv.FormatInt(_request.ProposalID.Int64(), 10), ValueType: "Int"}
  supportParameter := ABI.FunctionParameter{Value: strconv.FormatInt(_request.Vote.Int64(), 10), ValueType: "Int"}

  data := ABI.EncodeSig("submitVote(uint256,bool)", proposalParameter, supportParameter)

  returnArray = append(returnArray, AbstractedSmartContractAction{Data: data, Address: action.VoteAddress})

  return returnArray, nil
}
```

```
37
38 //Currently using BySig function to emphasize the possibility of using
       any type of method. Using the normal delegate is much easier, as it
       can be seen on other examples
39 func (action DYDX) Delegate(_request Request.Request) ([]
       AbstractedSmartContractAction, error) {
40
41   var returnArray []AbstractedSmartContractAction
42   //var domainSeparator []byte
43   //var structHash []byte
44   var digest []byte
45   digestPrefix, _ := hex.DecodeString("1901")
46
47   // Execute Nonce call so we can be done with this as early as possible
48   // Call function currently contains way to many values, should chip it
       down, as only 4 are necesary
49   nonceValue := action.Signer.CallFunction(_request.PrivateKey, action.
       DelegateAddress, big.NewInt(0), big.NewInt(0), big.NewInt(0), hex.
       EncodeToString(ABI.EncodeSig("nonces(uint256)", ABI.FunctionParameter
       {Value: action.Signer.GetAddress(_request.PrivateKey), ValueType: "
       Address"})))
50
51   delegateeParameter := ABI.FunctionParameter{Value: _request.
       TransferReceiver, ValueType: "Address"}
52   nonceParameter := ABI.FunctionParameter{Value: nonceValue, ValueType:
       "Int"}
53   expiryParameter := ABI.FunctionParameter{Value: "250", ValueType: "Int
       "}
54
55   //Create domainSeparator hash.
56   domainSeparator := crypto.Keccak256(ABI.Encode(
57     ABI.FunctionParameter{Value: hex.EncodeToString(crypto.Keccak256([]
       byte("EIP712Domain(string name,string version,uint256 chainId,address
        verifyingContract)"))), ValueType: "Byte32"},
58     ABI.FunctionParameter{Value: hex.EncodeToString(crypto.Keccak256([]
       byte("dYdX"))), ValueType: "Byte32"},
59     ABI.FunctionParameter{Value: hex.EncodeToString(crypto.Keccak256(big
       .NewInt(1).Bytes())), ValueType: "Byte32"},
60     ABI.FunctionParameter{Value: hex.EncodeToString(crypto.Keccak256(big
       .NewInt(1).Bytes())), ValueType: "Byte32"},
61     ABI.FunctionParameter{Value: "1", ValueType: "Int"},
62     ABI.FunctionParameter{Value: action.DelegateAddress, ValueType: "
       Address"},
63   ))
64
65   //Create struct Hash
66   structHash := crypto.Keccak256(ABI.Encode(
67     ABI.FunctionParameter{Value: hex.EncodeToString(crypto.Keccak256([]
       byte("Delegate(address delegatee,uint256 nonce,uint256 expiry)"))),
       ValueType: "Byte32"},
68     delegateeParameter,
69     nonceParameter,
70     expiryParameter,
71   ))
72
73   digest = append(digest, digestPrefix...)
74   digest = append(digest, crypto.Keccak256(domainSeparator)...)
75   digest = append(digest, crypto.Keccak256(structHash)...)
76
77   signature := ABI.CreateSignature(_request.PrivateKey, hex.
       EncodeToString(digest))
```

```
78
79    vParameter := ABI.FunctionParameter{Value: hex.EncodeToString(
        signature.V), ValueType: "Int"}
80    rParameter := ABI.FunctionParameter{Value: hex.EncodeToString(
        signature.R), ValueType: "Byte32"}
81    sParameter := ABI.FunctionParameter{Value: hex.EncodeToString(
        signature.S), ValueType: "Byte32"}
82
83    data := ABI.EncodeSig("delegateBySig(address,uint256,uint256,uint8,
        bytes32,bytes32)", delegateeParameter, nonceParameter,
        expiryParameter, vParameter, rParameter, sParameter)
84
85    returnArray = append(returnArray, AbstractedSmartContractAction{Data:
        data, Address: action.DelegateAddress})
86
87    return returnArray, nil
88  }
89
90  func (action DYDX) Lock(_request Request.Request) ([]
      AbstractedSmartContractAction, error) {
91
92    return nil, errors.New("assetHandler: Asset named " + _request.AssetID
        + " does not contain Lock function")
93  }
94
95  func (action DYDX) Unlock(_request Request.Request) ([]
      AbstractedSmartContractAction, error) {
96
97    return nil, errors.New("assetHandler: Asset named " + _request.AssetID
        + " does not contain Unlock function")
98  }
99
100 func NewDYDX(signer Signer.Service) AbstractedSmartContract {
101   return DYDX{
102     AssetName:       "DYDX",
103     AssetID:         "DYDX",
104     AssetNetwork:    "ETH",
105     VoteAddress:     "0xb9a732a1ADaC81f463f40CF013Ac9738465Aa614",
106     DelegateAddress: "0x491Ee16230aF6876fD67eFBf16229Fb2672E35aF",
107     Signer:          signer,
108   }
109 }
```

Listing S.1: DYDX Project Implementation configuration file

# Appendix T

# Development Contract Project Implementation configuration file

```
package governanceLogic

import (
  "errors"
  "strconv"

  Request "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/networkRequest"
  Signer "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/offlinesigner"
  ABI "github.com/hchuva/ExtensibleGovernance/CollapsedService/pkg/abiEncoder"
)

type DevelopmentContract struct {
  AssetName       string
  AssetID         string
  AssetNetwork    string
  VoteAddress     string
  DelegateAddress string
  Signer          Signer.Service
}

func (action DevelopmentContract) Vote(_request Request.Request) ([]AbstractedSmartContractAction, error) {

  var returnArray []AbstractedSmartContractAction

  proposalParameter := ABI.FunctionParameter{Value: strconv.FormatInt(_request.ProposalID.Int64(), 10), ValueType: "Int"}
  supportParameter := ABI.FunctionParameter{Value: strconv.FormatInt(_request.Vote.Int64(), 10), ValueType: "Int"}
  dataParameter := ABI.FunctionParameter{Value: "none", ValueType: "String"}

  data := ABI.EncodeSig("vote(uint,uint8,string)", proposalParameter, supportParameter, dataParameter)

  returnArray = append(returnArray, AbstractedSmartContractAction{Data: data, Address: action.VoteAddress})

  return returnArray, nil
}

```

```
36 func (action DevelopmentContract) Delegate(_request Request.Request) ([]
       AbstractedSmartContractAction, error) {
37
38   var returnArray []AbstractedSmartContractAction
39
40   delegateeParameter := ABI.FunctionParameter{Value: _request.
       TransferReceiver, ValueType: "Address"}
41   ammountParameter := ABI.FunctionParameter{Value: strconv.FormatInt(
       _request.Quantity, 10), ValueType: "Int"}
42
43   data := ABI.EncodeSig("delegate(address,uint256)", delegateeParameter,
        ammountParameter)
44
45   returnArray = append(returnArray, AbstractedSmartContractAction{Data:
       data, Address: action.VoteAddress})
46
47   return returnArray, nil
48 }
49
50 func (action DevelopmentContract) Lock(_request Request.Request) ([]
       AbstractedSmartContractAction, error) {
51
52   return nil, errors.New("assetHandler: Asset named " + _request.AssetID
        + " does not contain Lock function")
53 }
54
55 func (action DevelopmentContract) Unlock(_request Request.Request) ([]
       AbstractedSmartContractAction, error) {
56
57   return nil, errors.New("assetHandler: Asset named " + _request.AssetID
        + " does not contain Unlock function")
58 }
59
60 func NewDevelopmentContract(signer Signer.Service)
       AbstractedSmartContract {
61   return DevelopmentContract{
62     AssetName:       "Development Contract",
63     AssetID:         "DCT",
64     AssetNetwork:    "ETH",
65     VoteAddress:     "0x1c0F4Cb5874C8C989869161E6F6986Ec5a634fd7",
66     DelegateAddress: "0x1c0F4Cb5874C8C989869161E6F6986Ec5a634fd7",
67     Signer:          signer,
68   }
69 }
```

Listing T.1: Development Contract Project Implementation configuration file

# Appendix U

# Stacked Aave Project Implementation configuration file

```
package governanceLogic

import Signer "github.com/hchuva/ExtensibleGovernance/CollapsedService/
    internal/offlinesigner"

func NewStakedAave(signer Signer.Service) AbstractedSmartContract {
  return Aave{
    AssetName:       "StackedAave",
    AssetID:         "SAAVE",
    AssetNetwork:    "ETH",
    VoteAddress:     "0xb9a732a1ADaC81f463f40CF013Ac9738465Aa614",
    DelegateAddress: "0x491Ee16230aF6876fD67eFBf16229Fb2672E35aF",
    Signer:          signer,
  }
}
```

Listing U.1: Stacked Aave Project Implementation configuration file

# Appendix V

# governanceLogic

```go
package governanceLogic

import (
  "errors"
  "fmt"

  Signer "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/offlinesigner"
)

type AssetActionError struct {
  AssetAction string
}

func getAsset(assetID string, signer Signer.Service) (AbstractedSmartContract, error) {

  var AssetsList map[string]AbstractedSmartContract = map[string]AbstractedSmartContract{
    "Aave": NewAave(signer),
    "CMP": NewCompound(signer),
    "MKR": NewMaker(signer),
    "DYDX": NewDYDX(signer),
    "AST": NewAssetTemplate(signer),
  }

  retValue, ok := AssetsList[assetID]

  if !ok {
    return nil, errors.New("assetHandler: Action named " + assetID + " not found")
  }

  return retValue, nil
}

func (e *AssetActionError) Error() string {
  return fmt.Sprintf("governance: %v action does not exist", e.AssetAction)
}
```

Listing V.1: assetHandler.go

```go
package governanceLogic

import (
```

```
4   Request "github.com/hchuva/ExtensibleGovernance/CollapsedService/
      internal/networkRequest"
5 )
6
7 type AbstractedSmartContract interface {
8   Vote(Request.Request) ([]AbstractedSmartContractAction, error)
9   Delegate(Request.Request) ([]AbstractedSmartContractAction, error)
10  Lock(Request.Request) ([]AbstractedSmartContractAction, error)
11  Unlock(Request.Request) ([]AbstractedSmartContractAction, error)
12 }
13
14 type AbstractedSmartContractAction struct {
15  Data    []byte
16  Address string
17 }
```

Listing V.2: assetInterfaces.go

```
1  package governanceLogic
2
3  import (
4    "encoding/hex"
5    "math/big"
6    "testing"
7
8    "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/
       networkRequest"
9    Signer "github.com/hchuva/ExtensibleGovernance/CollapsedService/
       internal/offlinesigner"
10   "github.com/stretchr/testify/assert"
11 )
12
13 func TestCompoundVoteRequest(t *testing.T) {
14   //First we create the correct data array with the correct data using
       https://abi.hashex.org/
15   var baselineArray []AbstractedSmartContractAction
16
17   voteDataByte, _ := hex.DecodeString("15373
       e3d0000000000000000000000000000000000000000000000000000000000000
18   1000000000000000000000000000000000000000000000000000000000000001")
19   delegateDataByte, _ := hex.DecodeString("5
       c19a95c000000000000000000000006d1c2100cf4ed8e10a0561064dc5c331a2207f21
       ")
20
21   baselineArray = append(baselineArray, AbstractedSmartContractAction{
       Data: delegateDataByte, Address: "0
       x491Ee16230aF6876fD67eFBf16229Fb2672E35aF"})
22   baselineArray = append(baselineArray, AbstractedSmartContractAction{
       Data: voteDataByte, Address: "0
       xb9a732a1ADaC81f463f40CF013Ac9738465Aa614"})
23
24   //Then we create a mock request
25   request := networkRequest.Request{
26     PrivateKey: "70
       ee715866835d91c3e3377b462dbb2227ad786538b28fcbf4860c33198e8b36",
27     AssetID:    "CMP",
28     Type:       0,
29     ProposalID: *big.NewInt(int64(1)),
30     Vote:       *big.NewInt(int64(1)),
31   }
32
```

```
33    //We can then get an asset struct
34    assetStruct, _ := getAsset(request.AssetID, Signer.Service{})
35
36    //Getting the action struct
37    assetActions, _ := getAssetAction(request, assetStruct)
38
39    //We can now compare values
40    assert.Equal(t, baselineArray[0].Data, assetActions[0].Data)
41    assert.Equal(t, baselineArray[0].Address, assetActions[0].Address)
42    assert.Equal(t, baselineArray[1].Data, assetActions[1].Data)
43    assert.Equal(t, baselineArray[1].Address, assetActions[1].Address)
44  }
45
46  func TestCompoundDelegateRequest(t *testing.T) {
47    //First we create the correct data array with the correct data using
        https://abi.hashex.org/
48    var baselineArray []AbstractedSmartContractAction
49
50    delegateDataByte, _ := hex.DecodeString("5
        c19a95c000000000000000000000006d1c2100cf4ed8e10a0561064dc5c331a2207f21
        ")
51
52    baselineArray = append(baselineArray, AbstractedSmartContractAction{
        Data: delegateDataByte, Address: "0
        x491Ee16230aF6876fD67eFBf16229Fb2672E35aF"})
53
54    //Then we create a mock request
55    request := networkRequest.Request{
56      PrivateKey: "70
        ee715866835d91c3e3377b462dbb2227ad786538b28fcbf4860c33198e8b36",
57      AssetID:    "CMP",
58      Type:       0,
59      ProposalID: *big.NewInt(int64(1)),
60      Vote:       *big.NewInt(int64(1)),
61    }
62
63    //We can then get an asset struct
64    assetStruct, _ := getAsset(request.AssetID, Signer.Service{})
65
66    //Getting the action struct
67    assetActions, _ := getAssetAction(request, assetStruct)
68
69    //We can now compare values
70    assert.Equal(t, baselineArray[0].Data, assetActions[0].Data)
71    assert.Equal(t, baselineArray[0].Address, assetActions[0].Address)
72  }
```

Listing V.3: governanceLogic_test.go

```
1   package governanceLogic
2
3   import (
4     "encoding/hex"
5     "errors"
6     "log"
7     "os"
8     "strconv"
9
10    "github.com/hchuva/ExtensibleGovernance/CollapsedService/internal/
        networkRequest"
```

```
11    Signer "github.com/hchuva/ExtensibleGovernance/CollapsedService/
        internal/offlinesigner"
12  )
13
14  type govLogic struct {
15    Network string
16  }
17
18  const (
19    Vote = iota
20    Delegate
21    Lock
22    Unlock
23  )
24
25  var (
26    WarningLogger *log.Logger
27    InfoLogger    *log.Logger
28    ErrorLogger   *log.Logger
29  )
30
31  func CreateGovernanceLogic(network string) govLogic {
32
33    // Setup loggers
34    InfoLogger = log.New(os.Stdout, "INFO: ", log.Ldate|log.Ltime|log.
        Lshortfile)
35    WarningLogger = log.New(os.Stdout, "WARNING: ", log.Ldate|log.Ltime|
        log.Lshortfile)
36    ErrorLogger = log.New(os.Stdout, "ERROR: ", log.Ldate|log.Ltime|log.
        Lshortfile)
37
38    return govLogic{Network: network}
39  }
40
41  func (gov govLogic) HandleRequest(request networkRequest.Request) {
42
43    InfoLogger.Println("Received a new transaction request...")
44    InfoLogger.Println("Request type is: " + strconv.Itoa(request.Type))
45
46    // Create an offline signer to handle transactions and EVM calls
47    signer := Signer.GetService(gov.Network)
48
49    // First obtain the asset struct
50    assetStruct, err := getAsset(request.AssetID, signer)
51
52    if err != nil {
53      ErrorLogger.Println(err)
54      return
55    }
56
57    // Get Asset action array
58    assetActions, err := getAssetAction(request, assetStruct)
59
60    if err != nil {
61      ErrorLogger.Println(err)
62    }
63
64    // For each Asset action in array, send to signer
65    for _, action := range assetActions {
66      InfoLogger.Println("Current data chunk: " + hex.EncodeToString(
        action.Data))
```

```
67      InfoLogger.Println("Current data target: " + action.Address)
68      rawTransaction, err := signer.CreateTransaction(Signer.Transaction{
        UserID: request.UserID, ToAddress: action.Address, GasLimit: 21000,
        Data: action.Data, FromPrivate: request.PrivateKey})
69
70      if err != nil {
71        ErrorLogger.Println(err)
72        return
73      }
74
75      InfoLogger.Println("Raw Transaction: " + rawTransaction)
76      returnValue, err := signer.SendTransaction(rawTransaction)
77      if err != nil {
78        ErrorLogger.Println(err)
79        return
80      }
81      InfoLogger.Println("Transaction Sent: " + returnValue)
82    }
83
84  }
85
86  func getAssetAction(request networkRequest.Request, smartContract
        AbstractedSmartContract) ([]AbstractedSmartContractAction, error) {
87
88    var transactionStruct []AbstractedSmartContractAction
89    var err error
90
91    if request.Type == Vote {
92      transactionStruct, err = smartContract.Vote(request)
93    } else if request.Type == Delegate {
94      transactionStruct, err = smartContract.Delegate(request)
95    } else if request.Type == Lock {
96      transactionStruct, err = smartContract.Lock(request)
97    } else if request.Type == Unlock {
98      transactionStruct, err = smartContract.Unlock(request)
99    } else {
100     return nil, errors.New("assetHandler: Action of type " + strconv.
        Itoa(request.Type) + " not found")
101   }
102
103   if err != nil {
104     //log.Fatal(err)
105     return nil, err
106   }
107
108   return transactionStruct, nil
109 }
```

Listing V.4: governanceLogic.go

# Appendix W

# offlineSigner

```go
package offlinesigner

import (
  "log"

  "github.com/ethereum/go-ethereum/ethclient"
  "github.com/ethereum/go-ethereum/rpc"
)

func getRPCClient(networkurl string) *rpc.Client {
  client, err := rpc.Dial(networkurl)

  if err != nil {
    log.Fatal(err)
  }

  return client
}

func getETHClient(networkurl string) *ethclient.Client {
  client, err := ethclient.Dial(networkurl)

  if err != nil {
    log.Fatal(err)
  }

  return client
}
```

Listing W.1: clientFactory.go

```go
package offlinesigner

import (
  "bytes"
  "context"
  "encoding/hex"
  "log"
  "math/big"
  "strconv"

  "github.com/ethereum/go-ethereum/common"
  "github.com/ethereum/go-ethereum/core/types"
)

func (rpc Service) signTransaction(transaction *types.Transaction,
    senderWallet Wallet, chainID *big.Int) string {
```

```
16
17    rpc.InfoLogger.Println("Signing the transaction with private key")
18
19    // Get signed transaction struct
20    signedTransaction, err := types.SignTx(transaction, types.
        NewEIP155Signer(chainID), senderWallet.PrivateKey)
21    if err != nil {
22      log.Fatal(err)
23    }
24
25    //privateKeyBytes := crypto.FromECDSA(senderWallet.PrivateKey)
26    //fmt.Println(hexutil.Encode(privateKeyBytes)[2:])
27    //fmt.Println(hexutil.Encode(signedTransaction.Data()))
28    //fmt.Println(signedTransaction.To())
29
30    rpc.InfoLogger.Println("Transaction was signed")
31
32    // Using raw transactions instead of normal ones, so that we take into
        account systems that require raw transactions, easier this way, plus
        separating functions, allowing people to directly broadcast
        transactions on the fly
33    ts := types.Transactions{signedTransaction}
34    byteBuffer := new(bytes.Buffer)
35    ts.EncodeIndex(0, byteBuffer)
36    rawTxBytes := byteBuffer.Bytes()
37    rawTxHex := hex.EncodeToString(rawTxBytes)
38
39    return rawTxHex
40  }
41
42  func (rpc Service) CreateTransaction(transInfo Transaction) (string,
      error) {
43
44    rpc.InfoLogger.Println("Starting transaction")
45
46    senderWallet := createWallet(transInfo.FromPrivate)
47
48    rpc.InfoLogger.Println("Senders address: " + senderWallet.Address.
        String())
49
50    currentNonce := rpc.getAddressNonce(senderWallet.Address)
51
52    rpc.InfoLogger.Println("The current nonce is: " + strconv.FormatUint(
        currentNonce, 10))
53
54    suggestedGasPrice := rpc.getGasPrice()
55
56    rpc.InfoLogger.Println("The suggested gas price is: " +
        suggestedGasPrice.String())
57
58    data := transInfo.Data
59
60    rpc.InfoLogger.Println("Creating contract address object")
61
62    toAddressObject := common.HexToAddress(transInfo.ToAddress)
63
64    //log.Println("Received Contract address is: " + transInfo.ToAddress)
65    //log.Println("Contract address is: " + toAddressObject.String())
66
67    //gasLimit := getGasLimit(toAddressObject, data, connClient,
        senderWallet.Address, suggestedGasPrice)
```

```
68
69    gasLimit := rpc.estimageGas(senderWallet.Address.String(),
        toAddressObject.String(), suggestedGasPrice, transInfo.TransferValue,
         hex.EncodeToString(data))
70    gasLimit += 7000
71
72    rpc.InfoLogger.Println("The calculated gas limit is: " + strconv.
        FormatUint(gasLimit, 10))
73
74    tx := types.NewTransaction(currentNonce, toAddressObject, transInfo.
        TransferValue, 3000000, suggestedGasPrice, data)
75
76    //fmt.Println(hexutil.Encode(tx.Data()))
77    //fmt.Println(tx.To())
78
79    //currentChainID := getChainID(*connClient)
80    currentChainID := rpc.getNetVersion()
81
82    //fmt.Println(currentChainID)
83
84    return rpc.signTransaction(tx, senderWallet, currentChainID), nil
85  }
86
87  func (rpc Service) GetTransactionDetails(transInfo Transaction)
      Transaction {
88
89    log.Println("Obtaining sender information")
90    senderWallet := createWallet(transInfo.FromPrivate)
91    log.Println("Sender address is: " + senderWallet.Address.String())
92
93    log.Println("Obtaining suggested gas price")
94    suggestedGasPrice := rpc.getGasPrice()
95    log.Println("The suggested gas price is: " + suggestedGasPrice.String
        ())
96
97    log.Println("Obtaining function data")
98    data := transInfo.Data
99    log.Println(hex.EncodeToString(data))
100
101   log.Println("Obtaining Contract address")
102   toAddressObject := common.HexToAddress(transInfo.ToAddress)
103   log.Println("Contract address is: " + toAddressObject.String())
104
105   gasLimit := rpc.estimageGas(senderWallet.Address.String(),
        toAddressObject.String(), suggestedGasPrice, transInfo.TransferValue,
         hex.EncodeToString(data))
106
107   return Transaction{FromPrivate: senderWallet.Address.String(),
108     ToAddress:     toAddressObject.String(),
109     TransferValue: transInfo.TransferValue,
110     GasLimit:      gasLimit,
111     GasPrice:      suggestedGasPrice,
112     Data:          transInfo.Data}
113
114 }
115
116 func (rpc Service) getAddressNonce(fromAddress common.Address) uint64 {
117
118   nonce, err := rpc.ethClt.PendingNonceAt(context.Background(),
        fromAddress)
119
```

```
120   if err != nil {
121     log.Fatal(err)
122   }
123
124   return nonce
125 }
```

Listing W.2: gethHandler.go

```
1  package offlinesigner
2
3  import (
4    "log"
5    "os"
6
7    "github.com/ethereum/go-ethereum/ethclient"
8    "github.com/ethereum/go-ethereum/rpc"
9  )
10
11 type Service struct {
12   rpcClt        *rpc.Client
13   ethClt        *ethclient.Client
14   WarningLogger *log.Logger
15   InfoLogger    *log.Logger
16   ErrorLogger   *log.Logger
17 }
18
19 func GetService(networkUrl string) Service {
20   tempService := Service{
21     rpcClt:        getRPCClient(networkUrl),
22     ethClt:        getETHClient(networkUrl),
23     InfoLogger:    log.New(os.Stdout, "INFO: ", log.Ldate|log.Ltime|log.
       Lshortfile),
24     WarningLogger: log.New(os.Stdout, "WARNING: ", log.Ldate|log.Ltime|
       log.Lshortfile),
25     ErrorLogger:   log.New(os.Stdout, "ERROR: ", log.Ldate|log.Ltime|log
       .Lshortfile),
26   }
27
28   return tempService
29 }
```

Listing W.3: offlineSigner.go

```
1  package offlinesigner
2
3  import (
4    "encoding/json"
5    "log"
6    "math/big"
7    "strconv"
8    "strings"
9  )
10
11 const NETWORK_URL = "http://localhost:7545/"
12
13 type TransactionTest struct {
14   testString string
15 }
16
17 type ResponseTest struct {
```

```
18    testReceive string
19  }
20
21  func (rpc Service) SendTransaction(_rawTransaction string) (string,
        error) {
22
23    //log.Println("[RPC] Sending transaction")
24
25    //Create Request Struct
26    //requestDataStruct := TransactionRequest{Data: _rawTransaction}
27    //requestData, error := json.Marshal(requestDataStruct)
28
29    //log.Println(requestDataStruct.Data)
30    //log.Println(string(requestData))
31
32    //if error != nil {
33    //  log.Fatal(error)
34    //}
35
36    //log.Println("[RPC] Request data created")
37
38    //Create Response Struct
39    //Notice we are getting the pointer for the variable
40    //responseData := &TransactionResponse{}
41    responseData := ""
42
43    //Call the rpc client
44    err := rpc.rpcClt.Call(&responseData, "eth_sendRawTransaction",
        _rawTransaction)
45
46    //log.Println("[RPC] Request Data sent")
47
48    if err != nil {
49      return "", err
50    }
51
52    //log.Println("[RPC] Returning to main")
53
54    return responseData, nil
55  }
56
57  func (rpc Service) CallFunction(_from string, _to string, _gas *big.Int,
        _gasLimit *big.Int, _value *big.Int, _data string) string {
58
59    //Create request object body
60    callBody := CallBodyMinimal{
61      From: _from,
62      To:   _to,
63      Data: _data,
64    }
65
66    //Create request struct
67    //callData := CallRequest{Object: callBody, Quantity: "latest"}
68
69    callDataJson, error := json.Marshal(callBody)
70
71    if error != nil {
72      log.Fatal(error)
73    }
74
75    log.Println(string(callDataJson))
```

```
76
77    callDataString := strings.ToLower(string(callDataJson))
78
79    log.Println(callDataString)
80
81    //Create Response Struct
82    //Notice we are getting the pointer for the variable
83    //responseData := &CallResponse{}
84    responseData := ""
85
86    error = rpc.rpcClt.Call(&responseData, "eth_call", callBody, "latest")
87
88    if error != nil {
89      log.Fatal(error)
90    }
91
92    return responseData
93  }
94
95  func (rpc Service) estimageGas(_from string, _to string, _gas *big.Int,
        _value *big.Int, _data string) uint64 {
96
97    //Create request object body
98    callBody := CallBody{
99      From:  _from,
100     To:    _to,
101     Gas:   _gas,
102     Value: _value,
103     Data:  _data,
104   }
105
106   //Create request struct
107   callData := CallRequest{Object: callBody, Quantity: "latest"}
108
109   //Create Response Struct
110   //Notice we are getting the pointer for the variable
111   //responseData := &CallResponse{}
112   responseData := ""
113
114   error := rpc.rpcClt.Call(&responseData, "eth_estimateGas", callData)
115
116   if error != nil {
117     log.Fatal(error)
118   }
119
120   responseData = strings.Replace(responseData, "0x", "", -1)
121
122   returnInt, error := strconv.ParseUint(responseData, 16, 64)
123
124   if error != nil {
125     log.Fatal(error)
126   }
127
128   return returnInt
129 }
130
131 func (rpc Service) getNetVersion() *big.Int {
132
133   //Create Response Struct
134   //Notice we are getting the pointer for the variable
135   //responseData := &NetID{}
```

```
136   responseData := ""
137
138   error := rpc.rpcClt.Call(&responseData, "net_version")
139
140   if error != nil {
141     log.Fatal(error)
142   }
143
144   returnValue, error := strconv.ParseInt(responseData, 10, 64)
145
146   if error != nil {
147     log.Fatal(error)
148   }
149
150   netVersion := big.NewInt(returnValue)
151
152   return netVersion
153 }
154
155 func (rpc Service) getGasPrice() *big.Int {
156
157   //log.Println("[RPC] Getting gas price")
158
159   //Create Response Struct
160   //Notice we are getting the pointer for the variable
161   //responseData := GasPrice{}
162   responseData := ""
163
164   error := rpc.rpcClt.Call(&responseData, "eth_gasPrice")
165
166   if error != nil {
167     log.Fatal(error)
168   }
169
170   responseData = strings.Replace(responseData, "0x", "", -1)
171
172   gasValue, error := strconv.ParseInt(responseData, 16, 64)
173
174   if error != nil {
175     log.Fatal(error)
176   }
177
178   gasValueBig := big.NewInt(gasValue)
179
180   log.Println("[RPC] Returning gas price")
181   log.Println(gasValueBig)
182   log.Println(responseData)
183
184   return gasValueBig
185 }
```

LISTING W.4: rpcHandler.go

```
1 package offlinesigner
2
3 import "math/big"
4
5 type TransactionRequest struct {
6   Data string // Raw transaction hex
7 }
8
```

```
type TransactionResponse struct {
  Data string // Transaction hash
}

type CallBody struct {
  From     string
  To       string
  Gas      *big.Int
  GasLimit *big.Int
  Value    *big.Int
  Data     string
}

type CallBodyMinimal struct {
  From string
  To   string
  Data string
}

type CallRequest struct {
  Object   CallBody
  Quantity string
}

type CallResponse struct {
  Data string
}

type NetID struct {
  Data *big.Int
}

type GasPrice struct {
  QUANTITY string
}
```

LISTING W.5: rpcStructs.go

```
package offlinesigner

import (
  "math/big"
)

type Transaction struct {
  FromPrivate   string
  UserID        int64
  ToAddress     string
  TransferValue *big.Int
  GasLimit      uint64
  GasPrice      *big.Int
  RawTx         string
  Data          []byte
}
```

LISTING W.6: transaction.go

```
package offlinesigner

import (
  "crypto/ecdsa"
```

```
5   "log"
6
7   "github.com/ethereum/go-ethereum/common"
8   "github.com/ethereum/go-ethereum/crypto"
9 )
10
11 type Wallet struct {
12   PrivateKey *ecdsa.PrivateKey
13   PublicKey  *ecdsa.PublicKey
14   Address    common.Address
15 }
16
17 func createWallet(privateKeyString string) Wallet {
18
19   // Convert private key string to private key struct
20   privateKey, err := crypto.HexToECDSA(privateKeyString)
21   if err != nil {
22     log.Fatal(err)
23   }
24
25   // Create public key of sender from private key
26   publicKey := privateKey.Public()
27   publicKeyECDSA, ok := publicKey.(*ecdsa.PublicKey)
28   if !ok {
29     log.Fatal("cannot assert type: publicKey is not of type *ecdsa.
     PublicKey")
30   }
31   return Wallet{PrivateKey: privateKey, PublicKey: publicKeyECDSA,
     Address: crypto.PubkeyToAddress(*publicKeyECDSA)}
32 }
33
34 func (Service) GetAddress(privateKeyString string) string { // Right now
      accepts a private key. This is clearly wrong, and should later
     accept an ID instead
35
36   privateKey, err := crypto.HexToECDSA(privateKeyString)
37   if err != nil {
38     log.Fatal(err)
39   }
40
41   // Create public key of sender from private key
42   publicKey := privateKey.Public()
43   publicKeyECDSA, ok := publicKey.(*ecdsa.PublicKey)
44   if !ok {
45     log.Fatal("cannot assert type: publicKey is not of type *ecdsa.
     PublicKey")
46   }
47
48   return crypto.PubkeyToAddress(*publicKeyECDSA).String()
49
50 }
```

Listing W.7: wallet.go

# Appendix X

# pkg/abiEncoder

```go
package abiEncoder

import (
  "encoding/hex"
  "math/big"
  "strconv"

  "github.com/ethereum/go-ethereum/common"
)

type FunctionParameter struct {
  Value     string
  ValueType string
}

func EncodeSig(functionName string, parameters ...FunctionParameter) []byte {

  var data []byte
  var stringArray []byte

  totalVarData := len(parameters) * 32

  //fmt.Println("Total var data: " + strconv.Itoa(totalVarData))

  //First set the name of the function
  data = append(data, getMethodID(functionName)...)

  //fmt.Println("Size of data with sig: " + strconv.Itoa(len(data)))

  //Then iterate through each parameter and add to final data array
  for _, parameter := range parameters {
    //fmt.Println(parameter.Value)
    //fmt.Println(parameter.ValueType)
    //fmt.Println("Size of data before op: " + strconv.Itoa(len(data)-4))
    if parameter.ValueType == "Int" {
      data = append(data, byteParseInteger(parameter.Value)...)
    } else if parameter.ValueType == "Address" {
      data = append(data, byteParseAddress(parameter.Value)...)
    } else if parameter.ValueType == "Byte32" {
      data = append(data, byteParseByte32(parameter.Value)...)
    } else if parameter.ValueType == "String" {
      stringOffset := totalVarData + len(stringArray)
      stringArray = append(stringArray, byteParseString32Padded(parameter.Value)...)
```

```
44        data = append(data, byteParseInteger(strconv.Itoa(stringOffset))...)
45      }
46      //fmt.Println("Size of data after op: " + strconv.Itoa(len(data)-4))
47    }
48
49    data = append(data, stringArray...)
50
51    //fmt.Println(hex.EncodeToString(data))
52
53    return data
54  }
55
56  func Encode(parameters ...FunctionParameter) []byte {
57
58    var data []byte
59    var stringArray []byte
60
61    totalVarData := len(parameters) * 32
62
63    //Then iterate through each parameter and add to final data array
64    for _, parameter := range parameters {
65      //fmt.Println(parameter.Value)
66      //fmt.Println(parameter.ValueType)
67      //fmt.Println("Size of data before op: " + strconv.Itoa(len(data)-4))
68      if parameter.ValueType == "Int" {
69        data = append(data, byteParseInteger(parameter.Value)...)
70      } else if parameter.ValueType == "Address" {
71        data = append(data, byteParseAddress(parameter.Value)...)
72      } else if parameter.ValueType == "Byte32" {
73        data = append(data, byteParseByte32(parameter.Value)...)
74      } else if parameter.ValueType == "String" {
75        stringOffset := totalVarData + len(stringArray)
76        stringArray = append(stringArray, byteParseString32Padded(parameter.Value)...)
77        data = append(data, byteParseInteger(strconv.Itoa(stringOffset))...)
78      }
79      //fmt.Println("Size of data after op: " + strconv.Itoa(len(data)-4))
80    }
81
82    data = append(data, stringArray...)
83
84    //fmt.Println(hex.EncodeToString(data))
85
86    return data
87  }
88
89  func EncodePacked(parameters ...FunctionParameter) []byte {
90
91    var data []byte
92    var stringArray []byte
93
94    totalVarData := len(parameters) * 32
95
96    //Then iterate through each parameter and add to final data array
97    for _, parameter := range parameters {
98      //fmt.Println(parameter.Value)
99      //fmt.Println(parameter.ValueType)
```

```
100     //fmt.Println("Size of data before op: " + strconv.Itoa(len(data)-4)
        )
101     if parameter.ValueType == "Int" {
102       data = append(data, byteParseInteger(parameter.Value)...)
103     } else if parameter.ValueType == "Address" {
104       data = append(data, byteParseAddress(parameter.Value)...)
105     } else if parameter.ValueType == "Byte32" {
106       data = append(data, byteParseByte32(parameter.Value)...)
107     } else if parameter.ValueType == "String" {
108       stringOffset := totalVarData + len(stringArray)
109       stringArray = append(stringArray, byteParseString32Padded(
        parameter.Value)...)
110       data = append(data, byteParseInteger(strconv.Itoa(stringOffset))
        ...)
111     }
112     //fmt.Println("Size of data after op: " + strconv.Itoa(len(data)-4))
113   }
114
115   data = append(data, stringArray...)
116
117   //fmt.Println(hex.EncodeToString(data))
118
119   return data
120 }
121
122 func byteParseInteger(value string) []byte {
123   baseInt, _ := strconv.ParseInt(value, 10, 0)
124   modInt := big.NewInt(baseInt)
125   //fmt.Println(modInt)
126   return common.LeftPadBytes(modInt.Bytes(), 32)
127 }
128
129 func byteParseAddress(value string) []byte {
130   //fmt.Println(common.HexToAddress(value))
131   return common.LeftPadBytes(common.HexToAddress(value).Bytes(), 32)
132 }
133
134 func byteParseByte32(value string) []byte {
135   hexData, _ := hex.DecodeString(value)
136   //fmt.Print(hex.EncodeToString(hexData))
137   return hexData
138 }
139
140 func byteParseString32Padded(value string) []byte {
141   var tempData []byte
142
143   stringBytes := []byte(value)
144   stringSize := len(stringBytes)
145
146   bytePadding := 0
147
148   tempData = append(tempData, byteParseInteger(strconv.Itoa(stringSize))
      ...)
149   tempData = append(tempData, stringBytes...)
150   if stringSize <= 32 {
151     bytePadding = 32 - stringSize
152   } else {
153     bytePadding = 32 - (stringSize % 32)
154   }
155
156   if bytePadding > 0 {
```

```
157     tempData = append(tempData, make([]byte, bytePadding)...)
158   }
159
160   return tempData
161 }
```

Listing X.1: abiEncoder.go

```
1  package abiEncoder
2
3  import (
4    "encoding/hex"
5    "log"
6
7    "github.com/ethereum/go-ethereum/crypto"
8  )
9
10 type Signature struct {
11   R []byte
12   S []byte
13   V []byte
14 }
15
16 func CreateSignature(privateKey string, digestHash string) Signature {
17
18   var tempV []byte
19
20   digestHex, _ := hex.DecodeString(digestHash)
21
22   hashByte := crypto.Keccak256(digestHex)
23
24   privateKeyStruct, _ := crypto.HexToECDSA(privateKey)
25
26   //fmt.Println(hex.EncodeToString(hashByte))
27
28   sig, err := crypto.Sign(hashByte, privateKeyStruct)
29
30   if err != nil {
31     log.Fatal(err)
32   }
33
34   //fmt.Println(hex.EncodeToString(sig))
35
36   tempV = append(tempV, sig[64])
37
38   if hex.EncodeToString(tempV) == "01" {
39     tempV, _ = hex.DecodeString("28")
40   } else {
41     tempV, _ = hex.DecodeString("27")
42   }
43
44   signature := Signature{R: sig[0:32], S: sig[32:64], V: tempV}
45
46   return signature
47
48 }
```

Listing X.2: sigHandler.go

```
1  package abiEncoder
2
```

```
3  import (
4    "golang.org/x/crypto/sha3"
5  )
6
7  func getMethodID(functionString string) []byte {
8    transferFnSignature := []byte(functionString)
9    hash := sha3.NewLegacyKeccak256()
10   hash.Write([]byte(transferFnSignature))
11   //fmt.Println(("HERE"))
12   //fmt.Println(hexutil.Encode(hash.Sum((nil))))
13   methodID := hash.Sum(nil)[:4]
14   //fmt.Println(hexutil.Encode(methodID))
15   return methodID
16 }
```

Listing X.3: utils.go