School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

# Building Smart SAFEs

by

William Coulter

Thesis submitted as a requirement for the degree of

Bachelor of Engineering (Software)

Submitted: November 2021

Supervisor: Prof. Ron van der Meyden

Student ID: z5113817

Topic ID: Blockchain and Distributed Ledger Technology

# Abstract

With the technological advancements in blockchain and distributed ledger technology, there is opportunity to securely represent financial assets and agreements digitally without the need for a centralised authority.

A Simple Agreement for Future Equity (or a SAFE) is a legally enforceable financing contract between a company and investor. As the name suggests, the agreement is quite simple when compared to other financing contracts. The simplicity of the contract however, does not translate into implementating a SAFE contract on a distrbuted ledger technology such as blockchain, and there are many interesting problems to consider. That being said, SAFE contracts make a good starting point.

Ron van der Meyden and Michael Maher from the University of New South Wales have been analysing the applicability of blockchain platforms for equity financing contracts such as SAFEs in their draft paper *Can SAFEs be Smart?*.

This paper builds on the existing work by:

- Completing the designs for the full SAFE contract event cycle
- Implementing the designs in the Solidity programming language
- Analysing the appropriateness of a permissioned private-chain
- Creating a prototype that showcases how the platform could be used

This work also contributes towards the broarder goal of representing legal contracts in code by implementing the SAFE equity financing contract.

# Acknowledgements

I would like to thank my supervisor, Ron van der Meyden, for his help and guidance throughout this thesis. I appreciate the time you put aside for me each week and the quick and invaluable feedback and advice that you have provided me with.

This thesis was inspired by you and Michael Maher's work, who I would also like to thank, and could not have been possible without the both of you.

# Abbreviations

**SAFE** Simple Agreement for Future Equity

**IPO** Initial Public Offering

**ICO** Initial Coin Offering

**DAO** Decentralised Autonomous Organisation

**DAML** Digital Asset Modeling Language

**USA** United States of America

**GUI** Graphical User Interface

**IFRS** International Financial Reporting Standards

**ASX** Australian Stock exchange

**CHESS** Clearing House Electronic Subregister System

# Contents

# Chapter 1

# Introduction

Blockchain has been the advent for innovation and technological research. Since the release of Bitcoin in 2009 and Ethereum in 2014 there has been an explosion of companies looking to leverage blockchain, both legitimate and hype. In a 2018 survey conducted by PwC, out of 600 executives from 15 territories, 84% say their organisations have at least some involvement with blockchain [PwC18].

One of the main topics of blockchain technology has been how it can change the current ecosystem around equity financing - a method by which a company receives funding in exchange for its equity. One current financial instrument used to finance startups is the Simple Agreement for Future Equity (SAFE).

The goal of *Building Smart SAFEs* is to analyse the SAFE contract and propose ideas on how this can be represented in a blockchain application. This paper uses Ron van Der Meyden and Michael Maher's work in *Can SAFEs be Smart?* as a platform to build from.

# Chapter 2

# Background

The aim of this section to provide the reader with all the relevant information required to understand the content in the literature review and the work that was completed in this project.

## 2.1 Blockchain

Blockchain has become a popular topic of late with the explosion of interest in Bitcoin and Ethereum. The aim of this section is not to make the reader an expert on blockchain. Instead, this section will provide a high-level overview on what a blockchain is, will focus the type of blockchain this project will use and the tools associated with this.

The context of blockchain in this project will be introduced in section 2.1.4.

### 2.1.1 What

Although there are many different instances of blockchains such as Bitcoin, Ethereum and Cardano, a blockchain is a digital ledger of transactions that is distributed across an entire network of computer systems.

Computers in the blockchain are often referred to as "nodes". Participants in the blockchain can send their transaction to a node and have it recorded on that node's digital ledger. Since each node on the blockchain "hears" different transactions and has its own ledger, the nodes need some type of protocol to agree on which transactions are legitimate. This is known as *reaching consensus* [Fra20].

The consensus mechanism is where most blockchains differ and a lot of thought goes into ensuring nodes are incentivised to act honestly without having to trust a central authority.

### 2.1.2 Trustlessness

The concept of a system being able to operate without participants knowing and trusting each other, or a third party is known as *trustlessness* [Tac21].

A blockchain is said to be a trustless peer-to-peer network since:

- The consensus mechanism means there is no requirement for a trusted third party to operate the system.
- Cryptographic techniques means that participants can act securely and anonymously through the use of a public key, so participants have individual control over the visibility and operation of their transactions [SSD18].

Even though a blockchain can be referred to as trustless, it is more accurate to say that the trust in a blockchain is distributed onto the system that incentivises certain behaviours. Trust is minimised and placed onto the network rather than an individual, but is not eliminated entirely. This occurs through a *consensus mechanism* which varies between blockchains.

### 2.1.3 Decentralisation

Since nodes in a blockchain act independently and are not reliant on a central authority, the system is said to be *decentralised.*

Centralised systems require much more trust since participants delegate power to a central point, such as financial transactions being managed by a bank, business decisions being handled by a singular CEO or legal contracts being enforced by a government. In the case of the latter example, centralised systems can be desirable since people are happier placing their trust in an organisation of people rather than a system of code. Other advantages include clarity in decision-making, streamlined implementation of policies and control over the strategic direction of the organisation [Tex].

While centralised systems have their appeal, serious issues can emerge if the trusted third party can no longer be trusted. Trusted individuals in centralised organisations can become corrupt and act against the system for personal benefit (a solution to which is proposed in section 4.3). In addition, centralised systems are subject to system failures.

In the motivation section 3, there will be more discussion the advantages of decentralisation.

### 2.1.4 Digital Assets

So why is blockchain a topic of interest in a project that relates to legal equity financing contracts?

Blockchain has a huge use-case for digital representations of financial assets. Bitcoin is the most common example of a blockchain being used to represent a financial assets, namely the cryptocurrency "Bitcoin." According to the trading platform Plus500, Bitcoin is by far the most popular traded cryptocurrency [Plu].

But blockchain can be used for more than just representing a currency. In June 2019, the IFRS (an entity responsible for issuing accounting standards for the International Accounting Standards Board) published its decision on "Holdings of Cryptocurrencies" [PwC19]. Even though the decision states that holding a cryptocurrency does not inherently give rise to a contract between holder and another party, this does provide legitimacy to the role of blockchain in finance.

In addition, various companies have been investing time and money into representing digital assets on a blockchain such as the ASX's CHESS replacement, IBM's blockchain services and Facebook's Libra Coin

The particular type of financial assets involved in this project will be discussed in section 2.2 and onwards.

### 2.1.5 Ethereum, Smart Contracts and Solidity

Ethereum is another type of blockchain and is seen as a "2nd generation" blockchain since it has the capability to write and deploy *smart contract* [Rei20]. Note that Bitcoin also has smart contract capability however Ethereum smart contracts are much more expressive.

Smart contracts are pieces of code that execute on a blockchain to provide more expressive functionality[Fra21]. How much more expressive? According to Ethereum's LinkedIn page:

*Ethereum can be used to codify, decentralize, secure and trade just about anything*[Eth]

Whether this is true or not, the point is that this project is interested in Ethereum since it does allow for complicated structures such as a company cap table, investor and regulator to be represented on a blockchain.

Note that despite its name, the term *smart contract* is unrelated to *legal contracts* such as SAFEs. A smart contract acts like a object-orientated class which is essentially just a piece of code. I will attempt to make this distinction clear throughout this paper.

Ethereum smart contracts are written in the programming language *Solidity*.

### 2.1.6 Private Chains

Private chains are blockchains that allow a network of operators to restrict who can participate in the network and what access controls they are allowed [SHA19] at the expense of reducing trustlessness and adding centralisation.

While this type of governance goes against the ethos of cryptocurrency, there are legitimate business use-cases for a network like this.

Ethereum has multiple private chains known as Ethereum Enterprise.

## 2.2 A Startup: From Conception to IPO (or Acquisition)

Following the discussion on blockchain and its role in representing financial assets, this section will look at the specific type of financial assets that we want to analyse in this project.

A company's journey from idea conception to IPO or acquisition can be broken down as a series of capital raises. A capital raise is where founders of a company request money from investors in exchange for company shares.

The first of these capital raises is the seed raise which occurs once an idea has been fleshed out to the point where this can be explained in an enticing way to investors. This requires market research, an edge over competitors and sometimes beta trials of the product or service.

Following the seed raise, the company either becomes independently profitable or requests capital with further funding rounds known as “series A”, “series B” and “series C”. Assuming the company does not run out of money, eventually the company will be sold to the public in an IPO or be acquired by a larger company.

In 2019, 51.5% of companies failed within the first 2 years [Bry]. Because of this, the seed raise is very important and is often the most understood capital raise. This will be the primary focus of this document.

## 2.3 Initial Seed

Now that I have introduced the concept of an initial seed and explained its importance in the journey of a company, how does a company organise its initial seed?

This section will discuss 2 types of legal instruments used to execute an initial seed which are common in San Francisco: **convertible debt** and the **simple agreement for future equity** (SAFE).

### 2.3.1 Convertible Debt

A convertible debt agreement is a contract between a company and multiple investors that uses an instrument called a *convertible note* [Ral]. A convertible note is a loan that an investor lends to the company and has the following properties:

- **Principal Amount:** The amount the investor contributes.
- **Interest Rate:** The annual rate applied to the principal amount.
- **Maturity Date:** The time at which the principal amount and interest must be repaid.

The note provides the company with funds which it can use to achieve its goals. In exchange, the investor will receive interest and eventually this convertible note will *convert* into equity for the investors once the company undergoes its *equity round* (sometimes the investor also has the option to receive their principal amount back instead). Because of this, a convertible note can also have these properties:

- **Cap:** The maximum effective valuation that the owner of the note will pay to receive their equity, regardless of the company valuation at the time the note converts. If the company has a higher than expected valuation, this means the owner of the note can purchase equity in the company for less than other investors.
- **Discount:** An effective company valuation defined via a percentage off the round valuation.

The main disadvantage of convertible debt is that it requires purchasing *nominal debt* which typically has a lot of regulation and tax implications. Also, conversions at the equity round can be complicated and there is extra work for both parties if the contract needs to be extended.

### 2.3.2 SAFE

A SAFE is similar to convertible debt except that it has no principal amount, interest rate or maturity date. Since there is no maturity date in a SAFE, they can exist for as long as required by the company with minimal maintenance. SAFEs are also not debt instruments meaning they are not subject to as much regulation as the convertible note.

SAFEs (the first version being released in 2013 [Gra]) are the more modern legal contract and are desirable for startups for two main reasons:

1. A SAFE contract can be signed as soon as both parties are ready, being the founder and the investor. Since a SAFE has fewer properties than a convertible debt agreement, a SAFE is often faster to implement. This means startups can close deals with investors and receive funding sooner. This concept is known as *high resolution trading* [Con].
2. A SAFE contract is much simpler than convertible debt since there is no loan attached and there are typically fewer parties involved. This means that deals can be negotiated much faster saving time and money.

The SAFE has a trade-off between simplicity and comprehensiveness. While the SAFE addresses most of the common use-cases, there are always situations where a SAFE might not be the most appropriate contract.

### 2.3.3 SAFEs: A Further Note

There are 8 types of SAFEs in total that have been supported by Y-Combinator [Lev18].

SAFEs come in 2 forms either pre-money and post-money. This refers to the value of the company not including the seed raise money (pre-money) or including the latest capital injection (post-money). Currently, the post-money SAFE is the default supported SAFE by Y-Combinator.

Each form of SAFE has its own “flavour” depending on some parameters which a SAFE may or may not have. These parameters are a **Cap** and a **Discount** meaning there are 4 contracts per form of SAFE.

Why is this important background information? This project involves designing code that can accommodate all types of SAFE, so I’m mentioning the requirements of each type of SAFE now.

# Chapter 3

# Motivation

The aim of this chapter is to help the reader understand the reasoning behind this project and hopefully leave them inspired that the project's goals are interesting and important.

## 3.1 Why Digitise?

What are the benefits of digitising an equity contract? Why do we want to do this?

### 3.1.1 Automation

Through digitisation, rules constraining the issuance and transfer of equity rights can be automated and enforced. This means that investors receive a stronger guarantee of compliance with the rules.

### 3.1.2 Global Accessibility

SAFE contracts being implemented digitally also means that investors and founders have access to investment opportunities at any time of the day, not just typical business trading hours. This also means that the system can be accessed anywhere in the world, although since most equity contracts are legislated towards a particular country this does not provide a lot of benefit in practice.

### 3.1.3 The Bigger Picture

As mentioned previously, this project sits in a broader area of legitimising legally enforceable code.

Putting time and energy into implementing one legal contract digitally adds validity to the discussion around digital representations of other legal contracts. The challenges that are identified and overcome in this project all contribute towards a better understanding of how legal contracts in general can be digitised.

## 3.2 Why Digitise with Blockchain?

When it comes to digitisation, why would we want to do this on a distributed system such as Ethereum? Why not just implement the SAFE functionality on regular software architecture?

### 3.2.1 Decentralisation

The concept of decentralisation was brought up in section 2.1.3. This section describes the advantages of decentralisation in the context of equity financing.

No system is completely decentralised [Vos19]. Even a blockchain with many independent but equal network actors has a point of centralisation at the code written into its smart contracts or blockchain protocol. There are only different levels of decentralisation. Implementing a SAFE contract digitally on a distributed system will add levels of decentralisation which provide certain benefits such as:

- Avoidance of a single point of failure.
- Access to total ownership and control to the owner of the digital asset.

These benefits could not be realised or would have trade-offs if the system was not implemented on a distributed network.

### 3.2.2 Security

As discussed in section 2.1.2, even though all transactions on a public blockchain are visible to every participant, as discussed in section 2.1.6, Ethereum private chains can be used to limit who can access and view the network. Smart contract code can also be written to further enforce operations that various actors can perform on the network.

Additionally, since a blockchain is decentralised and consists of multiple nodes which can gracefully handle nodes dropping in and out of the network, this discourages any malicious attacks on a singular node from third parties.

While a blockchain does improve the security of the system, it has also introduced some different security risks. Since every node on the blockchain requires access to the smart contracts that execute on the protocol, bugs in smart contract code can be exploited. In addition, actors having their private key stolen can be a dangerous security risk.

### 3.2.3 Reduce Counterparty Risk

Transactions on a blockchain are performed atomically, meaning they are either accepted entirely by the system or not at all. These atomic operations will lead to a reduction in counterparty risks. An example of this is the atomic swap which involved Charlie Lee (the founder of Litecoin) performing a cross-chain atomic swap between Bitcoin and Litecoin [Lee17]. The atomic swap mitigates the dependency on an exchange platform.

### 3.2.4 Crowdfunding

Crowdfunding is the practice of funding a venture typically by raising small amounts of money from a large number of people via the Internet. SAFEs are being used in a lot of these crowdfunding rounds, making them even more pertinent for use on a blockchain.

# Chapter 4

# Literature Review

This project is about analysing the challenges and attempting to implement an Ethereum smart contract representation of a SAFE contract. With regards to this specific aim, there is not a lot of literature aside from Ron van der Meyden and Michael Maher's draft paper *Can SAFEs be Smart* and I will talk about this as my first item in this section.

A project like this however, does fit within the broader context of representing financial assets and legally enforceable assets on a blockchain. Because of this, in this section I will also discuss pieces of literature relating to using blockchain to implement a capital raise and on the limits of smart contracts to enforce the law.

## 4.1 Can SAFEs Be Smart?

This paper analyses the applicability of smart contract platforms for equity financing contracts such as SAFEs. *Can SAFEs Be Smart* is still being written by Ron van der Meyden and Michael Maher.

### 4.1.1 Overview

The paper is a very long document that looks at how a smart contract SAFE implementation can be architectured as well as an evaluation on how well smart contract languages such as Solidity and DAML can realise the requirements for this.

The paper also analyses security requirements, impediments to formalization, design patterns, privacy concerns and an evaluation.

### 4.1.2 Architecture for SAFE Smart Contracts

Out of this paper came another document called *Architecture for SAFE Smart Contracts*. This paper focuses on the implementation of a smart contract SAFE and proposes in detail a set of smart contracts that can be used to:

- Open a SAFE contract between an investor and a company. This SAFE can be pre-money or post-money and have various values for the **Cap** and **Discount** (if any).
- Handle the case of an **Equity Financing** event. The equity financing event is the happy case for the contract, i.e. when the company accepts payment from a venture capitalist on behalf of the investor (or the investor directly) in exchange for shares.

There is considerable design work and thought involved in supporting these requirements. Designs were written for representing a company and its components, an investor and the SAFE contract. There is also consideration for privacy and security - ensuring that each party has the appropriate access.

As mentioned, the equity financing event is the "happy case" once a SAFE contract has been established between a company and an investor. This involves the investor receiving equity in the company, typically by way of the company issuing more shares to provide the investor. There is also the possibility that the investor receives rights to purchase additional equity in the company at further equity rounds for a discounted price to maintain their current equity stake in the company (known as a Pro-Rata Rights Agreement) [Zegwn].

According to the SAFE contract documentation [Lev18], there are 2 other possible events:

- **Liquidity:** This can involve a merger with another company or if the company goes public.
- **Dissolution:** This can involve the company shutting down or going out of business.

## 4.2 ICOs

An ICO (Initial Coin Offering) is a common example of using blockchain to execute an type of equity round.

### 4.2.1 What

An ICO is an event where a company releases its own cryptocurrency (often referred to as a "token") with a purpose of exchanging this currency for funding [Mar19a]. This works by a founder starting a product and attaching a cryptocurrency with it which relates to the product. The founder will then ask for funding in exchange for the token which is assumed to be used a lot and eventually will be in demand, raising its value [She19]. This allows the founder to receive funding without giving up ownership of a company as opposed to other financing methods which exchange funding for equity in the company (see section 2.3.1 and 2.3.2).

### 4.2.2 IPO?

Even though the ICO was named after a similar, older equity model known as the IPO (Initial Public Offerings), there are many differences between the two.

In an IPO, a company's shares are released to the public which denotes a share ownership of that company. As discussed previously, this is not the case by default for ICOs, although there are some ICOs where the token in question also has company equity rights attached.

The primary difference between an ICO and an IPO is that an IPO is heavily regulated by the government whereas an IPO is relatively free from regulation. This makes ICOs quicker to release [Mar19b].

### 4.2.3 The Problem with ICOs

One of the fundamental problems with an ICO is that tokens sold do not usually make the founders responsible to the investor [ESwn]. Investors are not protected by governmental rules and are required to make their own due diligence and investment decision based off of the ICO's White paper.

While the lack of regulation and speed of deployment makes the ICO desirable, it also makes it an excellent tool for scammers. One report by the Satis Research Group shows that out of 1,500 surveyed ICOs, 78% of them were identified as scams which valued at around $1.3 billion USD [Gro18].

## 4.3 Decentralised Autonomous Organisations

### 4.3.1 The beginning of the DAO

One of the earliest instances of a Decentralised Autonomous Organization (DAO) was known as *The DAO* and was a form of investor-directed venture capital fund. It launched on the Ethereum network in April 2016 but was terminated in September due to users being able to exploit a vulnerability in the code [Sie16].

Although the initial project was short-lived, the principles and concepts used and its creation have been captured and applied elsewhere. Applications of these ideas are known as a *DAO* and is how the word will be used in this section.

### 4.3.2 Why DAOs?

Similar to the ICO, the DAO attempted to challenge the traditional governance structure existing in most companies. The issue with traditional structures was that it suffered from a single agent of the organisation being able to make a decision on behalf of another entity in the organisation. Common examples include a bar-manager deciding the uniforms of the bar-staff, a politician acting on behalf of their citizens or a CEO making a decision on behalf of shareholders. Most of the time this centralisation is desirable however as discussed in section 2.1.3, there is opportunity for the governing party to act in their own interest as opposed to the entity they are representing. Economists refer to this as the *principal-agent* dilemma [Vos19].

### 4.3.3 Structure

In traditional companies, agents are connected through employment contracts which are regulated by the law, so there is legal incentive to not disobey these contracts.

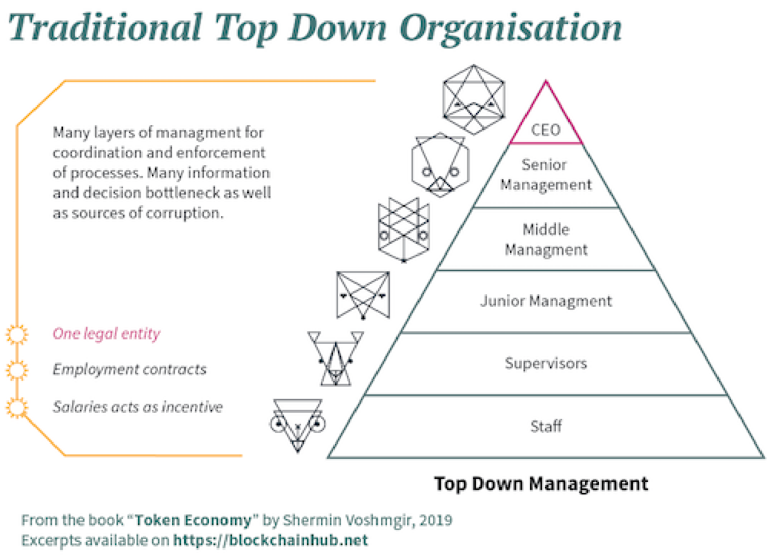

Traditional Top Down Organisation
Many layers of managment for coordination and enforcement of processes. Many information and decision bottleneck as well as sources of corruption.
One legal entity
Employment contracts
Salaries acts as incentive
CEO
Senior Management
Middle Managment
Junior Managment
Supervisors
Staff
Top Down Management
From the book "Token Economy" by Shermin Voshmgir, 2019
Excerpts available on https://blockchainhub.net

Figure 4.1: Source [Ber19b]

In DAOs, members are bound by incentives tied to the network tokens. The interests of all stakeholders are aligned by the consensus rules tied to the native token.

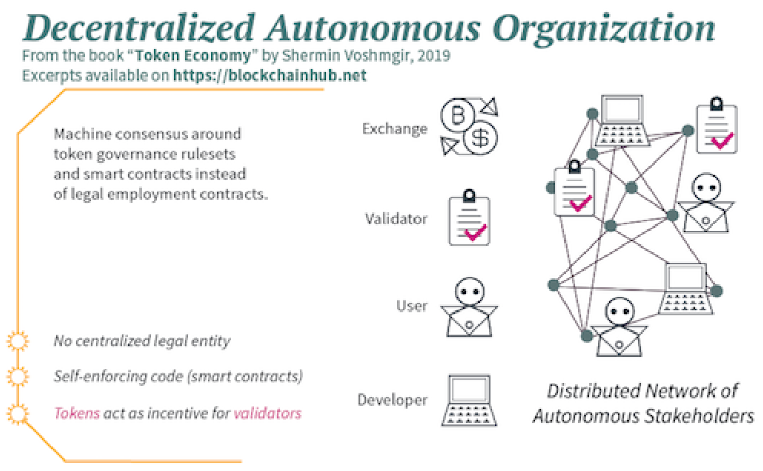

Decentralized Autonomous Organization
From the book "Token Economy" by Shermin Voshmgir, 2019
Excerpts available on https://blockchainhub.net
Machine consensus around token governance rulesets and smart contracts instead of legal employment contracts.
No centralized legal entity
Self-enforcing code (smart contracts)
Tokens act as incentive for validators
Exchange
Validator
User
Developer
Distributed Network of Autonomous Stakeholders

Figure 4.2: Source [Ber19a]

### 4.3.4 Relevance

DAOs are important to this project as they are another example of a blockchain implementation of performing an initial seed as a company. Although the first DAO is no longer used, the concepts involved in its creation are still being used and adapted upon.

This leads nicely into the next section where I discuss a company which tried to exist as a DAO-like structure.

## 4.4 A Real Life Company

Through my literature review, I had the opportunity to speak with founders of Geora who attempted to perform their initial seed through a DAO-like structure.

### 4.4.1 Geora

Geora is a company which attempts to use blockchain technology to improve the agricultural supply chain [Geo].

Being a company existing in the blockchain space, the founders are very aware of blockchain based equity models like the DAO and are more than comfortable with putting it to use.

### 4.4.2 The Foundation

Geora wanted to implement something they called the *Geora Foundation.*

Rather than attracting investors by exchanging money for company equity in the form of shares, investors could exchange their money for *tokens* in the Geora Foundation. These tokens provided owners with rights to receive dividends from the Geora company, as well as its intellectual property. All of this logic would be handled on the Ethereum blockchain.

Because this token structure is not a typical representation of any equity model, there was a lot of legal work put into validating an equity model like this. Additionally, the investors had to be believe that they could trust a Geora token as an asset, just like any regular share. This was ok, however, as the founders believed that security tokens were maturing in a way that they would be listed and regulated. This was not the case.

### 4.4.3 Why Stop the DAO?

According to the founders, most of the time spent with investors was used explaining what the Geora Foundation was and how it equates to equity for the investor, as opposed to talking about Geora and what its aims are. This lack of investor familiarity added another barrier when trying to land investors and the founders believed this was why it was quite difficult to receive funding during their initial seed round.

Geora is now ”Geora Proprietary Limited” which is a regularly listed Australian private company.

## 4.5 Limitations with Smart Contracts and Representing Legislation

This section will extract key points from the Harvard Law Article *An Introduction to Smart Contracts and Their Potential and Inherent Limitations* [LL18] which discusses the use of smart contracts in representing legal contracts.

### 4.5.1 Are Smart Contracts Enforceable?

One of the most pertinent issues when it comes to representing legal contracts in code is whether or not the code written in smart contracts has any legal weight behind it. This changes depending on the government but in the USA, a smart contract by itself is not enforceable by law. This makes sense when you consider the current text-based nature of legal contracts. A smart contract is code only so how could it be used to represent all of the text-based information contained in an agreement between two parties?

That being said, there are "ancillary smart contracts" which is a traditional text-based contract that references smart contracts to enforce certain provisions. These contracts are legally enforceable [LL18].

### 4.5.2 Amendments

At present, there is no simple way to amend a smart contract that has been deployed and is executing on a blockchain. Smart contracts can be programmed to accommodate for changing variables, but this all has to be prepared for and written in the code.

This poses challenges when it comes to legal contracts, since there is often unanticipated changes to agreements. Given the immutability of a blockchain, such a situation would require an amended smart contract to be deployed, and whatever state the old smart contract had captured, would have to be fed back into the new contract. [LL18]

### 4.5.3 Automation

One key function of smart contracts is their ability to automatically execute transactions. This is in contrast to current text-based contracts where humans meet and discuss whether the terms of the contract has been met and, if so transactions are executed. But what if there is a breach of the contract? Or what if there is a discussion moments before the anticipated execution of the contract where both parties agree to alter the terms slightly. Given that smart contracts are difficult to amend or terminate, there is likely to be no support for an *ad hoc* change to the contract. [LL18]

### 4.5.4 Representing Ambiguity

Often ambiguity in legal contracts is desirable. During the course of negotiations, parties engage in a cost-benefit analysis, knowing that the time they are spending doing so is a diminishing return. There may come a point where businesses do not want to spend more time calculating and estimating situations which may occur, but are very unlikely. Instead they like to leave the legalities of these situations ambiguous, and solve it later down the track if it occurs.

Since code is required to be objective, this ambiguity must be omitted from the smart contracts, making them less desirable than regular text-based contracts. [LL18]

### 4.5.5 On-chain vs Off-chain

Many smart contract implementations assume that the blockchain will receive information from resources outside of the network, known as ”off-chain” resources.

Consider an example where a weather based smart contract implementation requires wind speeds from various sources. These wind speeds are in constant flux and may be received by the blockchain's nodes at varying times. Considering that blockchains are required to reach a consensus (see section 2.1), the differing times that weather information is received in this application can make designing this application with blockchain quite challenging.

Blockchains actually have a solution to this problem known as Oracles, however this requires another party to consult when forming the legal contract [LL18].

# Chapter 5

# Problem Statement

The research question for this project is what are the challenges with implementing an equity financing contract on a blockchain application?

As explored in the literature review, there were some common issues faced by other attempts at creating a decentralised equity model for seed funding. These issues are:

1. **Non-compliance:** Some of these fundraising methods (in particular the ICO) were conducted in ways which violate laws governing equity financing.

2. **Investor Familiarity:** If an investor is not familiar with an equity model, this adds another layer of compilation when convincing them to invest.

3. **Representing law as code:** As explained in *An Introduction to Smart Contracts and Their Potential and Inherent Limitations* [LL18], there are many challenges with this.

4. **Reliance on a digital token:** As opposed to with receiving shares, investors had to believe that the digital tokens they received would translate into equity.

## 5.1 Why Digitise the SAFE?

One of the main reasons we want to digitise the SAFE is because it can overcome these issues discussed in the problem statement.

1. **Non-compliance:** The SAFE is already a legally compliant contract, however that is not to say that a digital implementation would be compliant. Assuming the legal contract is perfectly translated into code, regulators would still need access and visibility to the system.

2. **Investor Familiarity:** Investors are very familiar with the SAFE contract in some parts of the USA and in March 2021, Y-Combinator released side letters for companies formed in Canada, the Cayman Islands and Singapore [Lev18]. That being said, the SAFE is limited to these areas and is not a globally known equity financing model.

3. **Representing law as code:** This project serves as one implementation of a legal contract, which contributes to the issues of representing law as code in a small way.

4. **Reliance on a digital token:** This project does not solve this issue. There is an assumption that over time, investors will have more confidence in relying on a digital token as a legitimate asset.

## 5.2 Aims and Outcomes

Broadly, this thesis aims to design and implement an equity model on a blockchain that is conscious of the issues outlined above.

Specifically, this can be achieved by analysing the SAFE contract, building upon the existing designs described in *Architecture for SAFE Smart Contracts* and implementing it all on an Ethereum platform.

In an attempt to answer the research question, this project will involve the following contributions:

1. Write additional smart contract code to support a SAFE *Liquidity* and *Dissolution* events, building on existing work from *Architecture for SAFE Smart Contracts* [RvdM21a]. This should be flexible enough to support all the types of SAFEs discussed in section 2.3.3.
2. Model the permissioning requirements of a SAFE contract using Ethereum private chains (section 2.1.6).
3. Showcase the main features of the system in a UI prototype that connects to a private chain.

# Chapter 6

# Smart Contract Designs

All of the below designs are for a start-up in Australia using the **Postmoney SAFE No Discount** contract as specified by Y-Combinator [Lev18]. Note that even though these contracts are written specifically for the USA, in this project we assume that the SAFE contracts can be applied to Australian companies.

## 6.1 Liquidity Event

The liquidity event occurs when a *change of control* of the company happens. What this means is that another entity has become the beneficial owner of the company. For a private company in Australia this can occur in a few ways. In the context of a start-up, the most common reasons for a liquidity event are an acquisition or IPO [Hay21]. These are the two situations that will be considered in the designs below.

The full definition of a liquidity event has been included in the appendix A.3.

### 6.1.1 Starting a Liquidity Event

The controller of a company can start a liquidity event by calling the **start_liquidity_event** function on the **Safe_controller** smart contract.

```
function start_liquidity_event(
    address payable acquirer,
    uint256 expected_amount
) external OnlyController NoEventActive
```

This deploys a *Liquidity_event* smart contract which becomes the controller of the company and is used to handle the rest of the liquidity event. Once deployed, the *Safe_controller* acts as an interface to the *Liquidity_event* contract. The smart contract functions outlined in section 6.1.2 exist on both the *Safe_controller* and *Liquidity_event* contract. The controller of the company calls the function on the *Safe_controller* which performs some checks about the state of the network before calling the corresponding function on the *Liquidity_event* which handles all of the logic for the event. This interface pattern was used to isolate responsibilities between the *Safe_controller* (which ensures the smart contracts operate as specified by the SAFE legal contract) and the *Liquidity_event* (which ensures the Liquidity Event operates according to the steps outlined above).

This requires the address of the acquirer to be known and how much they are acquiring the company for in Ether. Note that this means the acquirer of the company needs to exist in the blockchain network. The acquirer can be an address to another smart contract which represents the acquiring company and performs additional logic once the acquisition is complete. This can be used to model more complicated conversions such as in an IPO.

Sometimes acquisitions can occur without a cash payment. For example, the acquiring company can pay for the acquisition using shares from their own company. These designs do not account for this situation and assume that all of the payments will occur as Ether.

It is assumed that all debts have been paid by the company before the acquisition occurs.

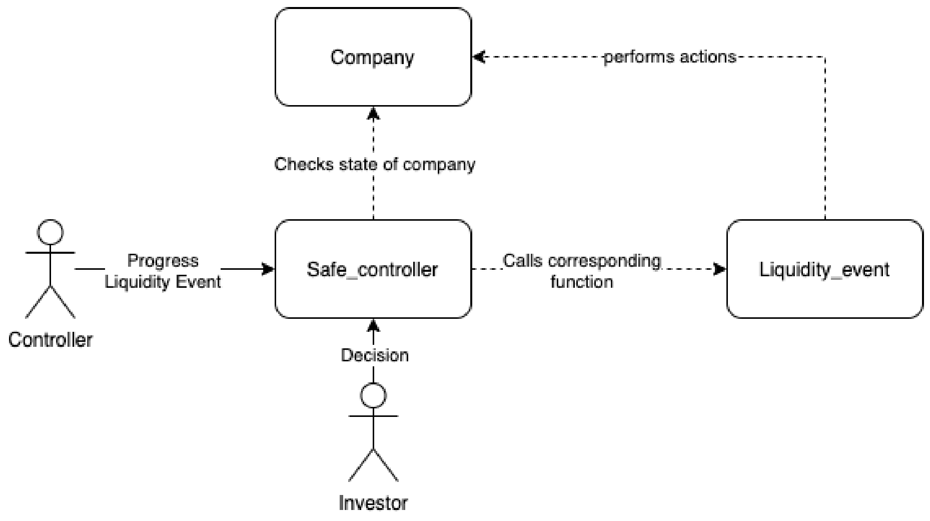


Figure 6.1: Interface design pattern

Once a liquidity event has begun, the *Safe_controller* will enter a *Liquidity_event_active* state. While in this state (as with the equity round event), only liquidity event related actions can be performed, meaning actions such as issuing a new SAFE or changing the company cap-table cannot occur. This means that the state of the company does not change during the acquisition process.

### 6.1.2 Liquidity Event Stages

Once a liquidity event has started, a sequence of stages are traversed. The sequence of stages for a liquidity event through acquisition has been designed as follows:

1. The common stock holders of the company approve the change of control.
2. The SAFE investors select their pay-out options.

3. The acquiring entity (another company or the public) deposits the funds that they are using to acquire the company.
4. The company makes payment to investors holding Preferred Stock and SAFE investors opting-in to the cash-out payment. [1] [2]
5. The company makes payment to investors holding Common Stock and SAFE investors opting-in to the conversion payment.
6. The company's Cap Table is updated to reflect the new ownership.

This event flow is captured in the enum **LiquidityStage**. The liquidity event smart contract maintains a state or **stage** of this enum.

```
enum LiquidityStage {
    Awaiting_approval,
    Awaiting_SAFE_payout_options,
    Awaiting_payment,
    Finalised,
    Aborted
}
```

Note that steps 4-6 occur atomically meaning once the acquiring entity has made payment for the acquisition (implying all other prior steps are also complete), all payments are made to participating entities control of the company is handed over to the acquirer. This is to provide the counter-party assurance discussed in section 3.2.3.

To move the liquidity event to the next stage, the controller can call the **progress_liquidity** function. Depending on the current stage of the liquidity event, this progression will

[1]As specified by the Liquidity Priority in the SAFE contract, prior to this step the company needs to make payment to all debt that it holds. Debt is not modelled in this project and so this will not be considered in this step.

[2]Note that Preferred Stock is not actually modelled in these designs. This is a realistic assumption since cap-tables in startups are typically simple.

revert or succeed. This manual behaviour was chosen since for some stages (such as *Awaiting_approval*), it is not clear when the event should progress. This also provides more autonomy to the controller of the contract.

At any point of the liquidity event (prior to the atomic operations in steps 4-6), the controller can abort the event with the **abort_liquidity_event** function, resetting the *Safe_Controller*'s status and setting the *Liquidity_Event* contract to *Aborted.*

### 6.1.3 Awaiting approval

Once a liquidity event has been initiated by the controller, the common stock holders of the company need to provide approval. Exactly how many common stock holders need to approve in order to validate the liquidity event can vary depending on a company's constitution. The *Liquidity_Event* contract in this project has been set up such that more than 50% of the common stock holders need to approve before the liquidity event can proceed to the next stage.

See the **finalise_approval** function in the *Liquidity_Event* smart contract for this implementation.

### 6.1.4 Awaiting SAFE payout options

In a liquidity event, a SAFE investor has the option to choose between a **Cash-out** amount or a **Conversion** amount. The value of the **Cash-out** amount is the principal that they paid for their SAFE. The value of the **Conversion** amount depends on other factors. One of these factors is the pay-out options of other SAFE investors.

Most of the time, one pay-out option will always result in a larger amount for the SAFE owner, regardless of the decisions of other SAFE investors. There are, however, some cases where the decisions of other SAFE investors *will affect* which pay-out option will result in a larger amount and calculating whether this case has risen can be done in polynomial time. Details on when this game-theory scenario occurs is discussed in section 7.

Because computations on a blockchain are expensive, establishing whether there is an ideal pay-out option for all SAFE investors and, if there is, what option the SAFE investor should choose is not implemented in the smart contracts. Instead, the *Awaiting_SAFE_payout_options* stage was created and SAFE investors must decide themselves in all situations.

#### 6.1.5 Awaiting payment

Once the payment has been received, steps 4-6 (that were mentioned earlier in section 6.1.2) are automatically executed. This is because this stage handles the transfer of Ether from the acquirer to the company and the transfer of company stock from the controller to the acquirer, and we want to ensure that the exchange is atomic.

See the **execute** function on the *Liquidity_Event* smart contract for implementation of the above logic.

Successful completion of this stage puts the *Liquidity_Event* smart contract in a *Finalised* stage.

## 6.2 Dissolution Event

A dissolution event is much simpler than a liquidity event. A dissolution event is any type of voluntary or involuntary wind up of the company. This can be because the founders of a company decide the company cannot be maintained or if pressure from the creditors demand the company to dissolve. In contrast to the liquidity event, no change of control occurs and the company will no longer be active.

The definition of the dissolution event outlined in the SAFE contract is in section A.4.

### 6.2.1 Starting a Dissolution Event

The controller of a company can start a dissolution event by calling the **start_dissolution_event** function on the **Safe_controller** smart contract.

```
function start_dissolution_event() external OnlyController NoEventActive
```

Notice that no arguments need to be passed to this function since no additional information needs to be known to start a dissolution event.

This deploys a *Dissolution_event* smart contract which, similarly to the way the liquidity event was structured, becomes the controller of the company and handles the rest of the dissolution event. See figure 6.1 for a diagram of this interface.

### 6.2.2 Dissolution Event Stages

The dissolution event has fewer stages to traverse:

1. The common stock holders of the company approve the dissolution.
2. The company makes payment to investors holding Preferred Stock and SAFE investors.
3. The company makes payment to investors holding Common Stock. [3]
4. The company smart contract is marked as inactive.

Similarly to the liquidity event, when a dissolution event is active, no other actions can be performed on the company. 2-4 occur atomically so once the dissolution event has been approved by a majority of common stock holders, the event cannot be aborted.

[3]Once again, the liquidity priority in the SAFE contract outlines the order in which participants are paid.

A similar **enum** type is also used to capture the various stages:

```
enum DissolutionStage {
    Awaiting_approval,
    Finalised,
    Aborted
}
```

### 6.2.3 Awaiting approval

Once a dissolution event has been initiated by the controller, the common stock holders of the company need to provide approval. Similarly with the liquidity event, there is an assumption that the company demands at least 50% of its common stock holders to approve the dissolution.

Once the dissolution has been approved, the rest of the dissolution event is executed. As outlined in steps 2-4, preferred stock holders and SAFE owners are paid at the same time. SAFE owners are paid their cash-out amount or are paid their principal at a pro-rata rate if the company does not have the equity to pay SAFE investors in full. Common stock holders are paid last, if there is any remaining equity in the company.

This puts the dissolution event smart contract into a *Finalised* state and marks the company as inactive. See the **finalise_approval** function on the *Dissolution_Event* contract for implementation.

# Chapter 7

# Liquidity Game Theory

In this section, I go into detail about how the **Conversion** amount is calculated in the case of a liquidity event. I also explain how the scenario can invoke game-theory strategy and touch on conclusions from Ron van Der Meyden's paper, *A Game Theoretic Analysis of Liquidity Events in Convertible Instruments* [vdM21].

## 7.1 Calculating the Conversion Amount

According to the Postmoney SAFE document [YCo21], the **Conversion Amount** is the price per share of Common Stock, multiplied by the Purchase Amount and divided by the Liquidity Price.

$$A_{conversion} = \boldsymbol{P_{common}} * \frac{\boldsymbol{A_{purchase}}}{\boldsymbol{P_{liquidity}}}$$

Where:

- $P_{common}$ is the price per share of Common Stock.
- $A_{purchase}$ is the Purchase Amount.

- $P_{liquidity}$ is the Liquidity Price.

The *Liquidity Price* is the price per share equal to the Post-Money Valuation Cap divided by the Liquidity Capitalisation, so:

$$A_{conversion} = P_{common} * \frac{A_{purchase}}{(\frac{\boldsymbol{Cap_{post-money}}}{\boldsymbol{C_{liquidity}}})}$$

Where:

- $Cap_{post-money}$ is the Post-Money Valuation Cap of the SAFE.
- $C_{liquidity}$ is the Liquidity Capitalisation.

The *Liquidity Capitalisation* is the total amount of stock existing in the company immediately prior to the Liquidity Event. This includes (without double-counting):

- All shares of Capital Stock issued and outstanding.
- All issued and outstanding Options and Promised Options (to the extent receiving Proceeds).
- All Converting Securities *excluding* convertible securities where the holders of the securities are receiving Cash-Out Amounts instead of Conversion Amounts.

Capital stock is the amount of common and preferred shares that a company has issued and recorded on their balance sheet [Kha21]. Options are not modelled in the smart contract designs so this will not be included in these equations.

The third item is interesting since it means that the Liquidity Capitalisation is dependent on the pay-out options of the SAFE owners, so:

$$A_{conversion} = P_{common} * \frac{A_{purchase}}{(\frac{Cap_{post-money}}{\boldsymbol{S_{capital}+S_{conversion}}})}$$

Where:

- $S_{capital}$ is the number of Capital Stock in the company (Preferred plus Common).
- $S_{conversion}$ is the number of stock being issued as a result of all the converting SAFEs.

$S_{conversion}$ can be thought of as a variable that changes depending on the decisions of other SAFE investors in the liquidity event. This means that $A_{conversion}$ cannot be calculated until all SAFE investors in the liquidity event have selected their pay-out option.

If we assume SAFE investors always desire a hirer amount, then SAFE investors in a liquidity event will choose the higher amount of $A_{cash}$ or $A_{conversion}$:

$$Decision = max\{A_{cash}, A_{conversion}\}$$

Except the value of $A_{conversion}$ *cannot be known* at the time the SAFE investor is making their decision.

As I will explain in the ensuing section, although the conversion amount cannot be known exactly, in most situations the question of which amount is larger between the cash and conversion amount is known.

## 7.2 Game Theory Appearance

Game theory is a branch of mathematics that deals with competitive situations where the outcome of a participants decision depends on the decisions of other participants [And21].

SAFE investors in a liquidity event can be thought of as a game where each player is trying to yield a better outcome for themselves (a higher pay-out amount) and the outcome of their decision is dependent on decisions of other SAFE investors (since the $A_{conversion}$ depends on the number of converting SAFEs in the liquidity event).

Note that $A_{conversion}$ will *decrease* when more SAFE investors decide to take a cash option because of the liquidity priority that means there is less equity in the company at the time the conversion amount is calculated.

A natural question to ask is whether there are some situations where it does not matter what other SAFE investors decide, the higher amount of the cash or conversion amount will be the same? For example, maybe even if all other SAFE investors choose a cash option, the conversion amount specified by your SAFE is still higher.

The above situation is said to have a unique best Nash Equilibrium, since there is a state where each player maximises their outcome and each player does not have to consider the decisions of other players in order to do so. Formally, in a Nash Equilibrium, no player has anything to gain by changing only their own strategy [Rub94]. If there is one unique decision for all players that achieve their optimal outcomes, this is considered a unique best Nash Equilibrium. In a true Game Theory scenario, there is no unique best Nash Equilibrium and the players must think strategically.

So when does a unique best Nash Equilibrium arise in the SAFE liquidity events? And if it exists, can this unique best Nash Equilibrium be calculated?

Ron van der Meyden formalised these ideas in his paper *A Game Theoretic Analysis of Liquidity Events in Convertible Instruments* [vdM21].

The conclusions were that, when SAFE contracts are uniformly one of the same type (e.g Post-Money, Valuation Cap, No Discount), a unique best Nash Equilibrium does exist and it can be calculated in polynomial time.

# Chapter 8

# Blockchain Network

Following on from the smart contract designs, this chapter discusses the blockchain network that the smart contracts will be deployed to with a focus on how the permissioning requirements are realised.

## 8.1 Hyperledger Besu

Hyperledger Besu is an Ethereum-based blockchain designed for enterprise-friendly applications. Besu can be used for private, permissioned networks (recall section 2.1.6) and was the blockchain client used in this project to deploy smart contracts on.

### 8.1.1 Permissioning in Besu

Permissioning on a blockchain network is a method of restricting nodes and accounts that can access the network, and what parts of the network they can access.

Permissioning with Besu can be targeted at two different entities [Bes20]:

- **Node**: Restrict which nodes can join the network and what each node can see.
- **Account**: Restrict which accounts can send transactions to the network and what each account can see.

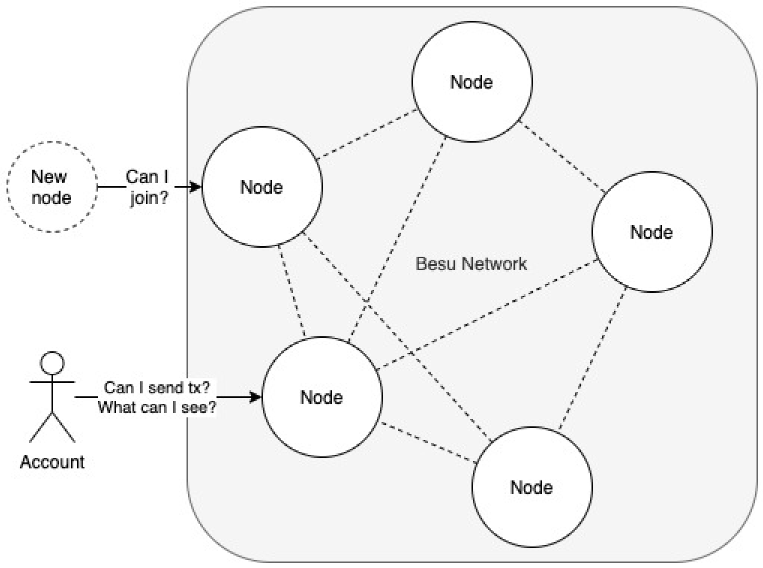


Figure 8.1: Besu rejecting access to nodes and accounts

The way in which the above permissions are implemented can occur at two parts of the network:

- **Local**: Occurring at the node level where each node has a configuration file that it uses to control permissions. Each node has a copy of the same configuration file and Besu provides an admin API to permeate files amongst the nodes. The configuration file in this project remains static so this type of management is not used.
- **On-chain**: Occurring through smart contracts on the network that control permissions.

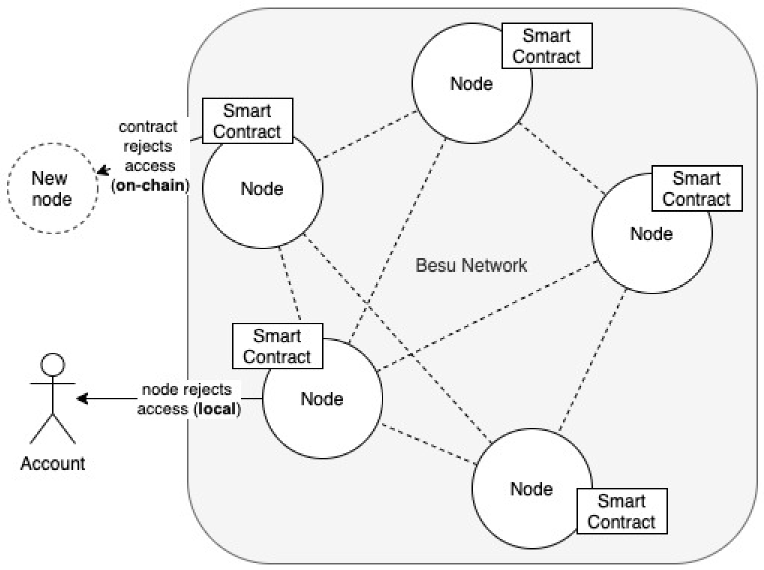


Figure 8.2: Besu permissioning being implemented locally and on-chain

This project uses a combination of these methods to restrict access to nodes and accounts.

### 8.1.2 Consensus Mechanism

Besu offers multiple consensus mechanisms (see section 2.1.2 for more on consensus mechanisms) that its nodes can use to agree on the state of the network. The consensus mechanism chosen for this project is a proof of authority called IBFT 2.0. IBFT 2.0 was initially proposed by Roberto Saltini and David Hyland-Wood in their paper *IBFT 2.0: A Safe and Live Variation of the IBFT Blockchain Consensus Protocol for Eventually Synchronous Networks* [SHW19] which built on the existing IBFT protocol.

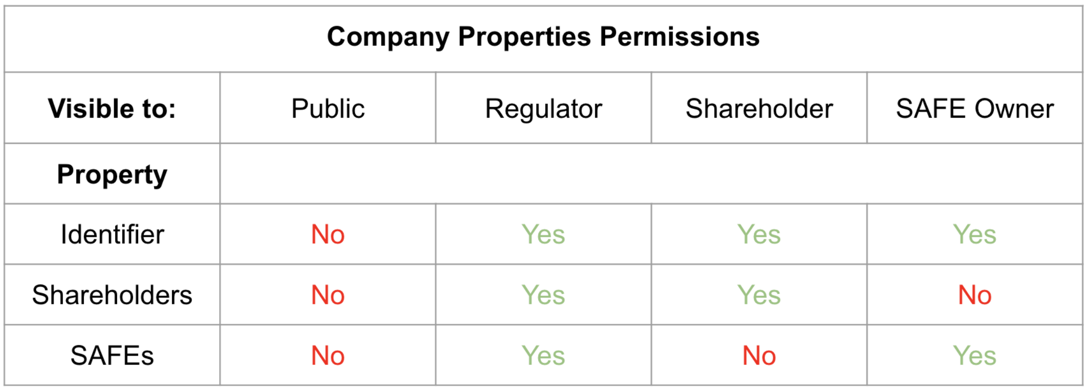
In proof of authority, the network has a notion of a *validator* node which validate transactions and blocks and take turns in creating the next block. The reason IBFT 2.0 was selected for this project is because it ensures immediate finality and is robust as an *eventually synchronous* network. This means transactions will have a guaranteed order once they appear on chain and their latency is somewhat bound. These features are desired when designing a network that is dealing with company information.

Note that IBFT 2.0 is not the only consensus mechanism that offers these features but it the one that Besu provides. See [Bes21c] for more information on how Besu implements IBFT 2.0.

## 8.2 Implementation

Following on from the Besu information, this section discusses the permission model that this project attempted to mimic and how this was accomplished with Besu.

### 8.2.1 The Goal



**Company Properties Permissions**

| **Visible to:** | Public | Regulator | Shareholder | SAFE Owner |
|---|---|---|---|---|
| **Property** | | | | |
| Identifier | No | Yes | Yes | Yes |
| Shareholders | No | Yes | Yes | No |
| SAFEs | No | Yes | No | Yes |

Figure 8.3: Permission Model Goal

Regulatory restrictions vary with jurisdiction, company size, whether the company is public or private and even with what type of assets it holds. In this project, the network was designed for a start-up in Australia since these are typically the types of companies that use SAFE contracts.

The permissioning in this section has been set up such that more complex permissioning situations can be implemented, however actually implementing these is a potential extension on this project.

Figure 8.3 shows the permission model that was implemented in this network. A row indicates a property of the company that exists as data stored on chain and a column represents a *role* in the network who can access this part of the company. Note that there is a role that is not shown in this table, being the *company controller*. The controller is represented by an Ethereum address (meaning it can be a user or another smart contract) and is the only actor who can perform updates to the company's information. For example, adding a new shareholder, offering a SAFE or setting a new controller. The controller of a company has permission to view all company information.

### 8.2.2 Restricting Node Access

Validators in an IBFT 2.0 blockchain can view all parts of the network since they have to receive and process transactions, as well as store a history of the blockchain locally. Because of this, it is important that not anyone can join the network as a validator, else company information will be totally available to the public [1]. To implement this, the nodes of the network are targeted with *local* permissioning as discussed in section 8.1.1.

Besu lets users specify which nodes can access the network as validators by adding the *enode url* to the **node-allowlist** in the permissions configuration file of each node [Bes21a]. This can be specified in the genesis file for the network - see A.2 for actual implementation. Besu provides API methods for updating the **node-allowList** of each node [Bes21b] however this feature was not explored in this project and the network is assumed to be a static set of validators.

[1]Note that private companies *can* actually reveal parts of their company to the public. This project assumes companies will not do this.

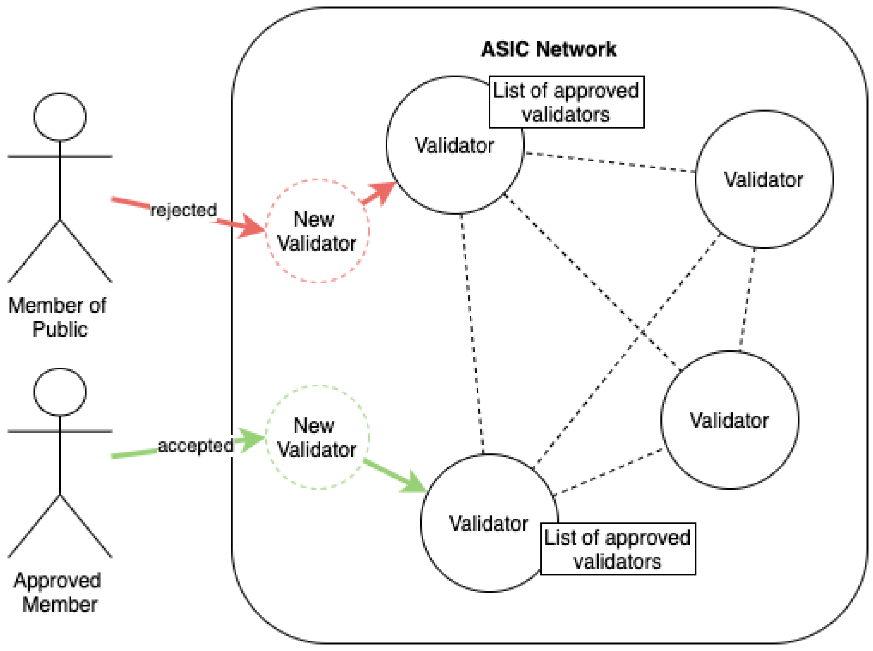


Figure 8.4: Private Chain Restriction Access

Since validators can access all parts of the network, it is assumed that the validators are trusted parties who have an interest in maintaining the network. Figure 8.4 shows a private chain that a regulatory body like *ASIC* is running.

### 8.2.3 Compromised Node

This type of **local** permissioning is relatively resilient to malicious attacks. Consider a situation where a node is compromised in the network. Due to the IBFT 2.0 consensus mechanism, if this compromised node tried to approve malicious transactions, this would disagree with the majority of other nodes in the network (IBFT 2.0 requires at least 4 nodes) and so the data of the network maintains its integrity. This will remain the case as long as a majority of the nodes in the network remain "healthy" and uncompromised. Despite this, the attacker will have access to view all information that is being sent to the network.

### 8.2.4 Restricting Account Access

So now that members of the public are restricted from joining the network as a validator at a *node* level, how are members of the public restricted from joining at an *account* level and how are various types of accounts restricted from accessing information in a company? This is where Besu's *on-chain* permissioning discussed in section 8.1.1 is used.

The on-chain permissioning essentially means that the permissioning is stored on the blockchain in the form of a smart contract. As opposed to local permissioning which is stored at the node level, on-chain permissioning will have its history maintained as it becomes a part of the blockchain's ledger. This provides a more decentralised method of permissioning and allows the network's permission requirements to be updated by deploying new smart contracts and migrating the data [2]

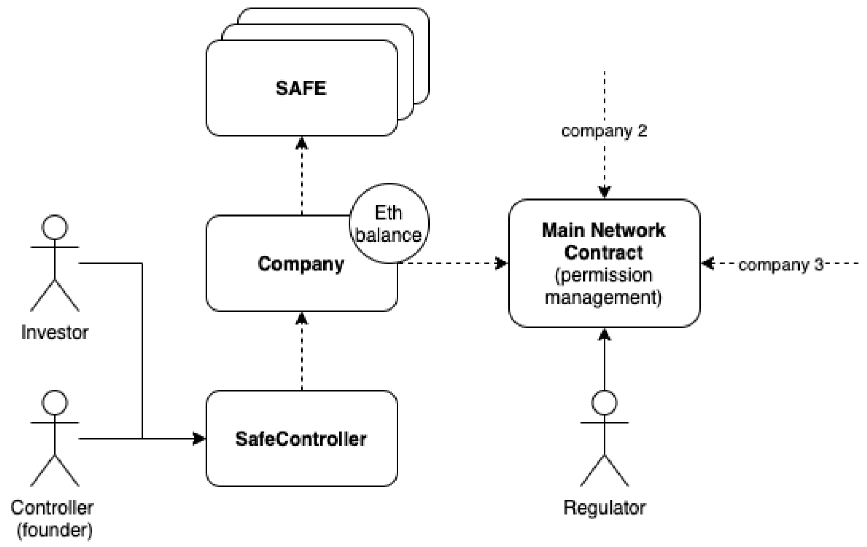


Figure 8.5: On-chain Permissioned Architecture

[2]Blockchain data migration can actually be really difficult if the network has not been set up to accommodate for this. See `https://research.csiro.au/blockchainpatterns/general-patterns/migration-patterns/` for a list of blockchain migration patterns proposed by the CSIRO.

Figure 8.5 introduces the idea of a **Main Network Contract**. This is a smart contract that has been implemented in the *RolesManager.sol* file - see A.1 for full implementation. The *RolesManager* contract defines a list of **Roles** that can exist in the network, which mimic what was defined in figure 8.3.

```
enum Role {
    Regulator,
    Shareholder,
    SafeOwner
}
```

The *RolesManager* contract maintains a list of *regulators* which can only be edited by other regulators in the network. This information is only stored on the smart contract, there is no information encoded into a transaction that contains information on the role of the sender.

```
function checkRole(
    Role[] memory roles,
    address sender
) external OnlyCompany returns (bool)
```

The *RolesManager* contract exposes a **checkRole** function which takes a list of **roles**, and an address **sender** and returns **true** if the address is one of the roles in the company that is requesting the information.

The **checkRole** method is then used in the *Company* smart contract to limit who can access what information in the company, based on their role. For example, the modifier **CompanyParticipants** restricts access to a function on the company based on whether or not the sender is a shareholder or SAFE owner in the company.

```
    modifier CompanyParticipants() {
        Role[] memory roles = new Role[](3);
        roles[0] = Role.Regulator;
        roles[1] = Role.Shareholder;
        roles[2] = Role.SafeOwner;

        require(
            rolesManager.checkRole(roles, msg.sender) ||
                // We also allow the controller to act
                msg.sender == controller,
            "Company: sender is not a company participant"
        );
        _;
    }
```

This is just one modifier that can be written with the **checkRole** function that the *RolesManager* exposes, further granularity can be provided by extending the number of roles defined in the contract and writing new modifiers in the *company* contract.

Other modifiers were written in the *Company* contract and place on certain functions in order to achieve the permission behaviour specified in figure 8.3. See the full company contract in A.1.

In order for a company to join the network, the must be approved by a regulator.

```
    function registerCompany(address c)
        external
        OnlyRegulator
        returns (address)
```

The **registerCompany** function deploys a new *Company* contract and sets the company controller as the provided argument **c**. This *Company* contract has pre-written permissions that assert permissioning according to the network's requirements. This ensures that users cannot join the network with their own written smart contract and bypass the permissioning that the network requires.

Since **registerCompany** deploys its own Company contract, this means users cannot specify their own logic in a smart contract and register it on the network, instead they have to use the pre-written contracts. Designing a means to provide flexibility in the company contracts without compromising the permissioning requirements outlined in figure 8.3 is a point of extension in this project.

# Chapter 9

# The Prototype

This chapter describes the prototype that was built in this project. The prototype consists of a GUI which is connected to a locally running instance of an Ethereum chain. The purpose of the prototype is to showcase all of the work that has been done in chapters 6 and 8 and understand how these can be applied in a business application.

The prototype is designed to support three roles in the network:

1. **Regulator**: An actor responsible for managing the network and controlling which companies can join. This encoded as a **Regulator** in the **Role** enum described in section 8.2.4.
2. **Investor**: An investor is considered to be a shareholder or safe-owner in a company. This is encoded as a **Shareholder** or **SafeOwner** in the **Role** enum described in section 8.2.4.
3. **Company Controller**: The listed controller of the company. Typically this is also an investor in the company, however since the controller of a company will be performing very different actions to an investor, this was separated into its own section of the prototype.

The GUI was built using React (`https://reactjs.org/`) which is a javascript framework for building user interfaces. Ethers (`https://docs.ethers.io/v5/`) is a javascript library that provides a useful abstraction layer over common Ethereum methods such as generating, signing and sending transactions to a set of smart contracts. Typescript (`https://www.typescriptlang.org/`) was also used on top of everything to provide extra compile-time assurance.

This chapter will present screenshots of the GUI that are relevant to each section. For a full list of screenshots, see appendix A.5.

## 9.1 Accessing the GUI

The GUI sends transactions to the Besu Ethereum private chain and therefore must sign messages on the behalf of a user's account. Metamask (`https://metamask.io/`) is a crypto wallet that can be used to store private keys and interact with blockchains. Metamask is used in this project for the prototype since it is a common tool for decentralised apps and does not store any user credentials on a hosted server - all private information remains on the user's local machine. Metamask has a Google Chrome extension (`https://metamask.io/download.html`) that is assumed to be installed when interacting with the GUI.

If the prototype does not detect a connected Ethereum account through Metamask, a screen will show to prompt a connection to Metamask (figure 9.1):

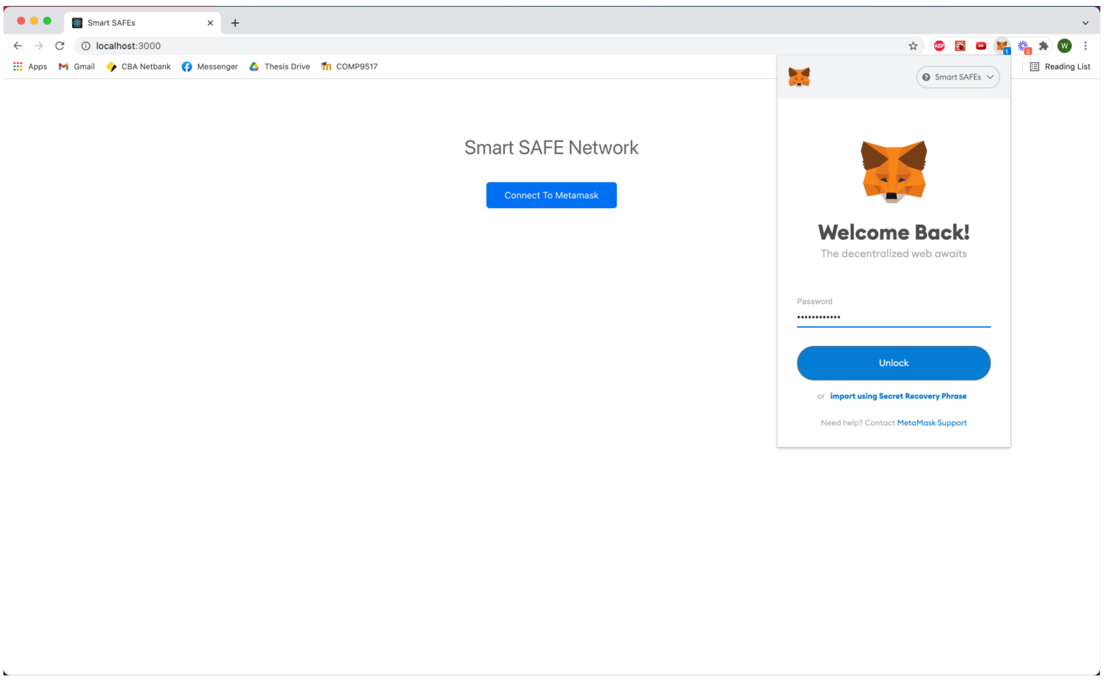


Figure 9.1: Connecting to the prototype with Metamask Screen

Metamask's Chrome extension stores the account details in the **window.ethereum** global variable. If this variable is undefined, the Metamask Chrome extension is not installed on the client's browser. Otherwise, the GUI prompts the user to log in and extracts information about the client's connection:

```
if (window.ethereum) {
    await window.ethereum.enable(); // waits for Metamask login

    const provider = new ethers.providers.Web3Provider(window.ethereum);
    const signer = provider.getSigner();
    const address = await signer.getAddress();
    const balance = await provider.getBalance(address);
    return { provider, signer, address, balance };
}
```

The **provider** is a variable that represents a connection to a node (read more here: https://web3py.readthedocs.io/en/stable/providers.html) and is how the GUI sends transactions to the locally running Ethereum blockchain. The **signer** is a variable specific to the logged in Ethereum account. This variable is used to sign transactions before sending them to the blockchain using the **provider**.

Once connected, the user is taken to a screen which prompts them to specify the role that they are joining network as. To join as a **Regulator**, the address of the *RolesManager* smart contract needs to be supplied. The prototype will then perform a check on the supplied smart contract to confirm that the connected account is in fact a regulator in the network. The security and authentication involved in this is discussed further in section 9.3. To join the network as an **investor** or **controller**, the address of the company smart contract needs to be supplied. Once again, checks are performed on the supplied smart contract to ensure that the connected account is in fact an investor or controller of the company.

Note that the fact that investors and controllers supply the address of the company they are accessing means that the screens only support viewing information on one company at a time. If an investor wants to view their investments in another company, they must reconnect via the ”select role” screen and provide a different company address. This is marked as future work in section 9.4.

## 9.2 GUI Functionality

The GUI does not capture all functionality that is available in the smart contracts. It is a convenience tool for interacting with the smart contracts designed to showcase the main functionality. This section discusses what this functionality is and what each role in the network can achieve through this prototype.

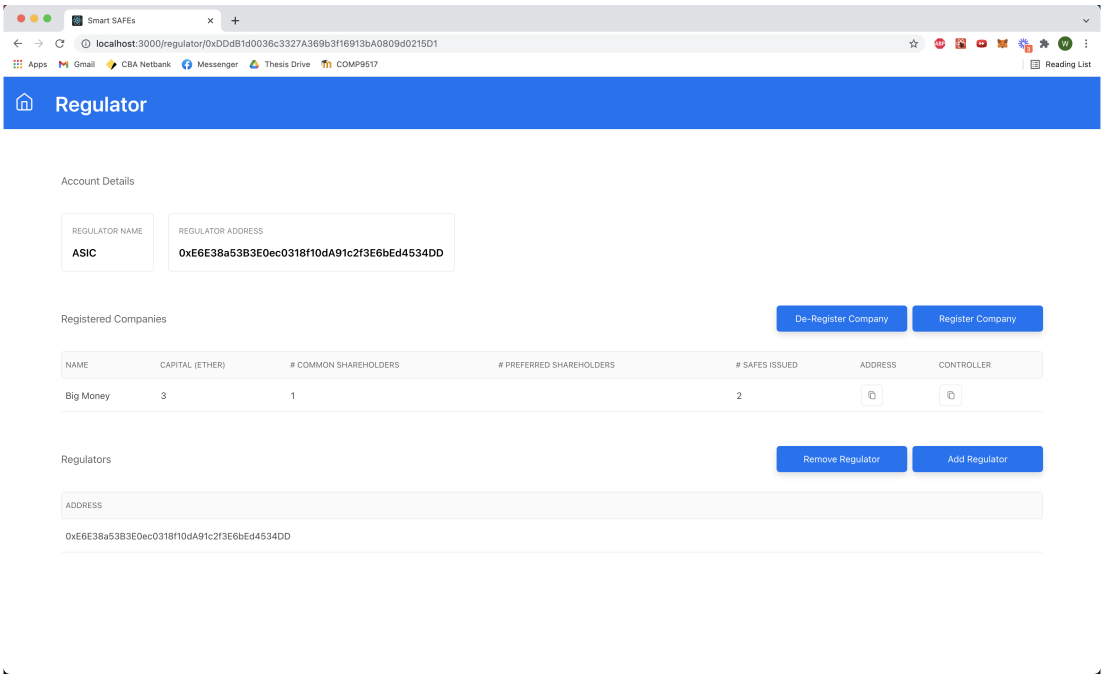
An example of a functionality that is not captured in the GUI is making payments as a company controller.

```
function payment(uint256 amount, address payable recipient)
    public
    OnlyController
```

This is exposed as a function on the *Company* smart contract however there is no part of the GUI that enables a controller to call this function. If a controller wants to call this function, they will have to sign and send their own transaction manually to the Ethereum chain.

### 9.2.1 Regulator

For a regulator, the prototype acts as a very simple management tool.



Account Details

| REGULATOR NAME | REGULATOR ADDRESS |
|---|---|
| ASIC | 0xE6E38a53B3E0ec0318f10dA91c2f3E6bEd4534DD |

Registered Companies

| NAME | CAPITAL (ETHER) | # COMMON SHAREHOLDERS | # PREFERRED SHAREHOLDERS | # SAFES ISSUED | ADDRESS | CONTROLLER |
|---|---|---|---|---|---|---|
| Big Money | 3 | 1 | | 2 | | |

Regulators

| ADDRESS |
|---|
| 0xE6E38a53B3E0ec0318f10dA91c2f3E6bEd4534DD |

Figure 9.2: Regulator screen

Figure 9.2 shows an authenticated regulator screen. From here, a regulator has a view of all the companies and regulators in the network. The regulator is also able to add and remove new companies and regulators.

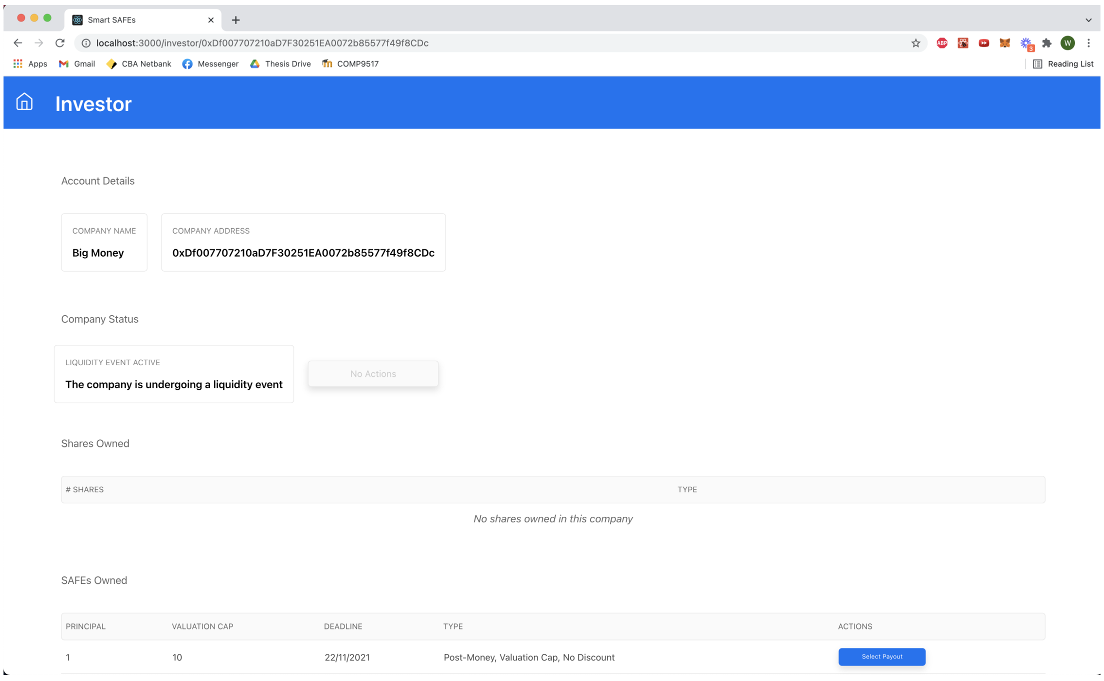
The "Register Company" button makes a call to the **registerCompany** function discussed in section 8.2.4. Prior to this step, there is an assumption that the regulator has had a discussion with founder of the company to establish and complete any regulatory requirements. This discussion occurs off-chain and is not modelled by this network.

If a company is de-registered from the network, this does not destroy the *Company* smart contract that represents the de-registered company however the company contract is removed from the network's list of registered companies and is essentially locked from performing any actions. This is implemented due to the fact that company smart contracts defer requests they receive to the *RolesManager* smart contract to confirm that such an action is permitted, as shown in figure 8.5. An action is permitted if it is compliant with the network permission requirements (shown in figure 8.3) and if the company is a registered company. The fact that the company is no longer registered means actions will be rejected.

There is more thought that can be put into this. For example, if a company not compliant and needs to be de-registered, the company might have debt obligations to its investors that it needs to fulfil once de-registered. As it stands now, the company controller of a de-registered company cannot perform such actions and a regulator would be required to perform this.

### 9.2.2 Investor

For an investor, the prototype will show any shares or SAFEs that the investor has in the company. Depending on the state of the company, the actions that an investor can perform will change as well.



Investor

Account Details

| COMPANY NAME | COMPANY ADDRESS |
|---|---|
| Big Money | 0xDf007707210aD7F30251EA0072b85577f49f8CDc |

Company Status

LIQUIDITY EVENT ACTIVE
The company is undergoing a liquidity event

No Actions

Shares Owned

| # SHARES | TYPE |
|---|---|

*No shares owned in this company*

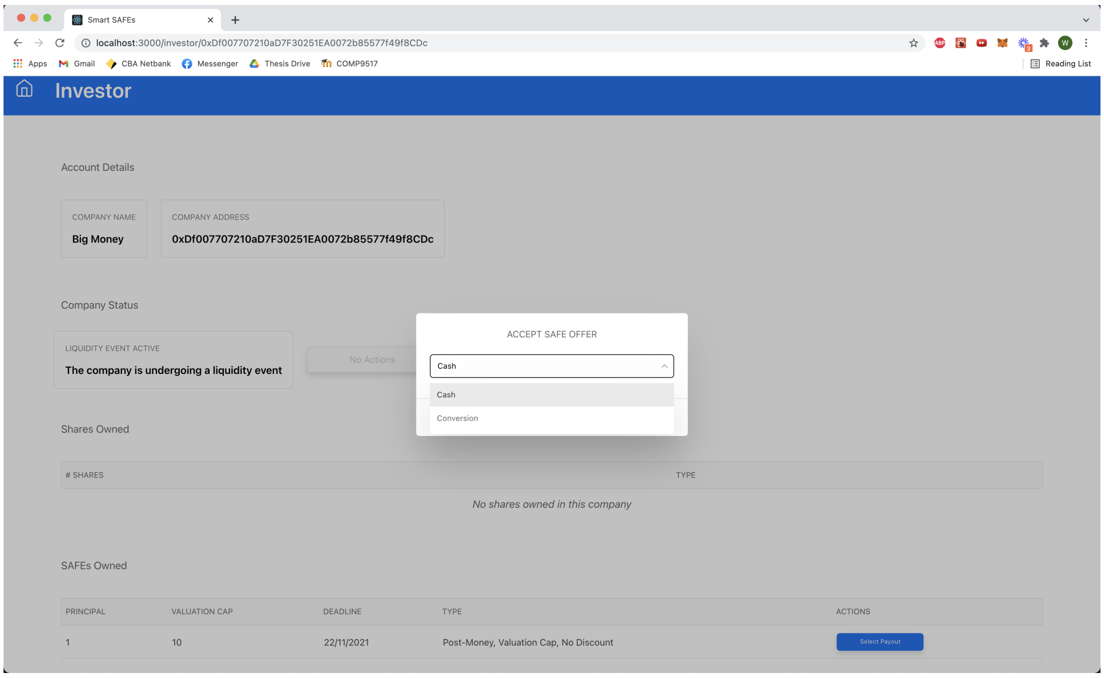
SAFEs Owned

| PRINCIPAL | VALUATION CAP | DEADLINE | TYPE | ACTIONS |
|---|---|---|---|---|
| 1 | 10 | 22/11/2021 | Post-Money, Valuation Cap, No Discount | Select Payout |

Figure 9.3: Investor screen

Figure 9.3 shows the view of an investor who owns a SAFE in the company *Big Money* undergoing a liquidity event. Next to the *Company Status*, there is a disabled button that states *No Actions*. If this investor was a common stock holder in this company, this button would provide the investor an option to *approve* or *disapprove* the liquidity event.

In the *SAFEs Owned* section, the investor has the option to select their payout option. This brings up a modal and is shown in figure 9.4.



Account Details

| COMPANY NAME | COMPANY ADDRESS |
|---|---|
| Big Money | 0xDf007707210aD7F30251EA0072b85577f49f8CDc |

Company Status

LIQUIDITY EVENT ACTIVE
The company is undergoing a liquidity event

No Actions

ACCEPT SAFE OFFER
Cash
Cash
Conversion

Shares Owned

| # SHARES | TYPE |
|---|---|

*No shares owned in this company*

SAFEs Owned

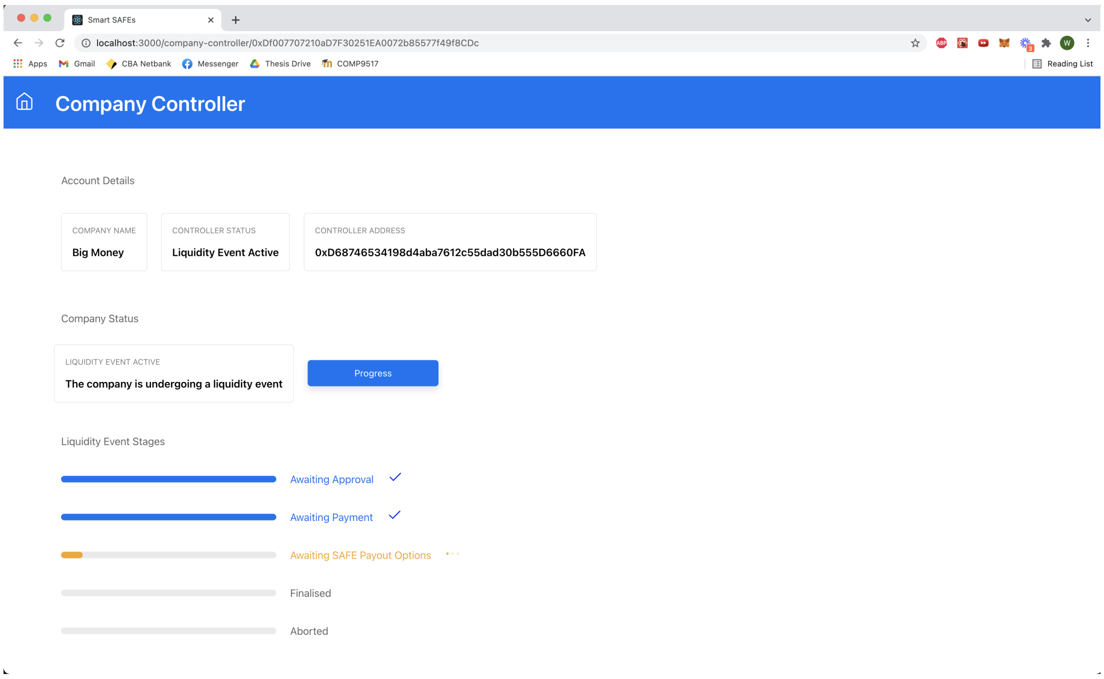
| PRINCIPAL | VALUATION CAP | DEADLINE | TYPE | ACTIONS |
|---|---|---|---|---|
| 1 | 10 | 22/11/2021 | Post-Money, Valuation Cap, No Discount | Select Payout |

Figure 9.4: SAFE Liquidity Payout Option

### 9.2.3 Company Controller

To authenticate the controller of a company, recall from figure 6.1 that while SAFEs are active in the company (prior to an equity round or liquidation event), all of the interaction between the controller and the company usually occurs through the *Safe_controller* contract to ensure that the system behaves in accordance with the SAFE legal contract. Because of this, the company controller screen will first check the immediate controller listed on the company. If this immediate controller is not the actor connected to the prototype, the prototype will then check the controller of the immediate controller of the company. So even though the listed *controller* of the company smart contract is the address of the *Safe_controller* contract, as long as the connected account is the listed controller of the *Safe_controller*, the prototype will grant access to this screen and perform all further actions through the *Safe_controller* contract.

This is a limitation since it only supports a stack of controllers of one depth. There is a use-case for supporting very complex controller hierarchies however since this project focuses on the management of SAFE investments in a company, such a controller structure has not been considered.


Smart SAFEs
localhost:3000/company-controller/0xDf007707210aD7F30251EA0072b85577f49f8CDc
Apps
Gmail
CBA Netbank
Messenger
Thesis Drive
COMP9517
Reading List

Company Controller

Account Details

COMPANY NAME
Big Money

CONTROLLER STATUS
Liquidity Event Active

CONTROLLER ADDRESS
0xD68746534198d4aba7612c55dad30b555D6660FA

Company Status

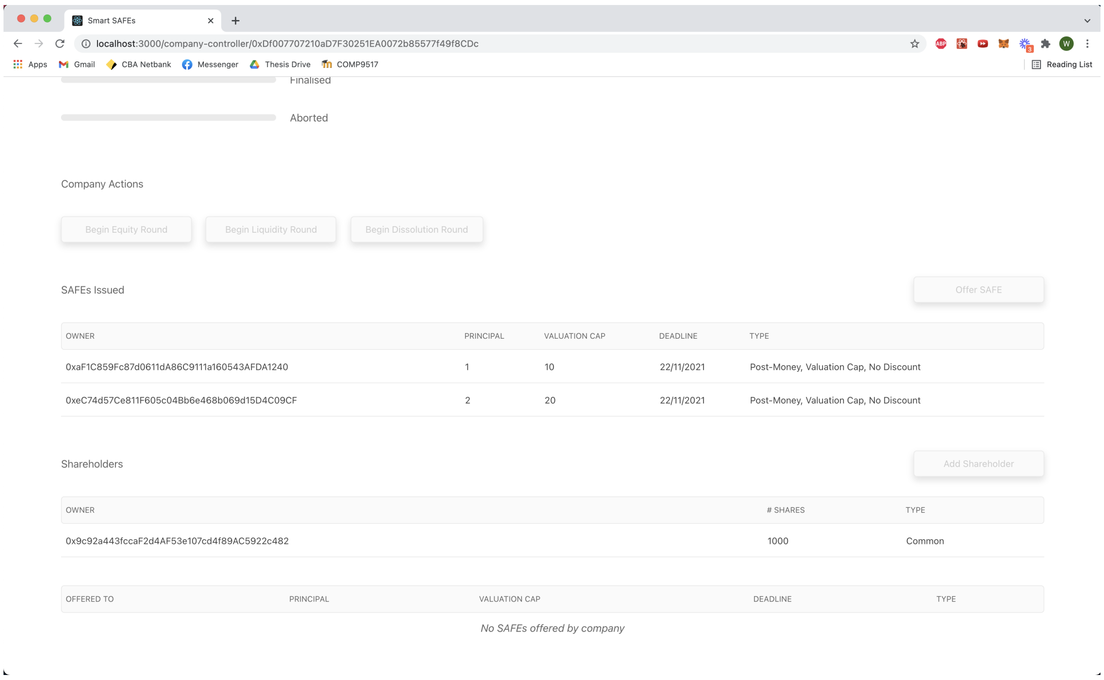
LIQUIDITY EVENT ACTIVE
The company is undergoing a liquidity event

Progress

Liquidity Event Stages

Awaiting Approval ✓
Awaiting Payment ✓
Awaiting SAFE Payout Options
Finalised
Aborted

Figure 9.5: Company Controller Screen



Company Actions

SAFEs Issued

| OWNER | PRINCIPAL | VALUATION CAP | DEADLINE | TYPE |
|---|---|---|---|---|
| 0xaF1C859Fc87d0611dA86C9111a160543AFDA1240 | 1 | 10 | 22/11/2021 | Post-Money, Valuation Cap, No Discount |
| 0xeC74d57Ce811F605c04Bb6e468b069d15D4C09CF | 2 | 20 | 22/11/2021 | Post-Money, Valuation Cap, No Discount |

Shareholders

| OWNER | # SHARES | TYPE |
|---|---|---|
| 0x9c92a443fccaF2d4AF53e107cd4f89AC5922c482 | 1000 | Common |

| OFFERED TO | PRINCIPAL | VALUATION CAP | DEADLINE | TYPE |
|---|---|---|---|---|

*No SAFEs offered by company*

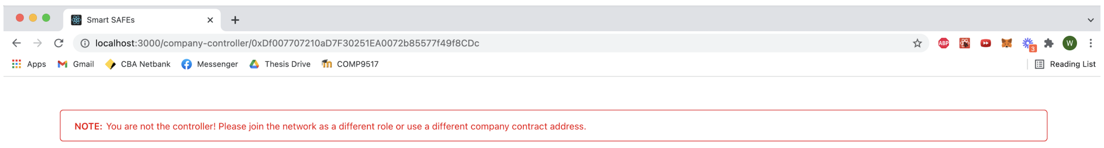
Figure 9.6: Company Controller Screen (scrolled down)

As shown in figure 9.5 and 9.6, the prototype is aware of the company state, being *Awaiting SAFE Payout Options* of a Liquidity Event. Notice that the controller cannot perform certain actions on the company while in this state, such as starting an equity round, offering a new SAFE or adding a shareholder. Similar to how security is handled in section 9.3, all event logic is also handled on the smart contract level so even if a user bypasses this part of the prototype and tries to do something like change the cap-table of the company during an acquisition (an action that could be used to mislead an acquirer), the transactions will be rejected by the network.

In this screen, the controller of a company can view some basic information about the SAFEs it has offered as well the shareholders.

## 9.3 Security

When an account tries to access a section of the prototype, the UI authenticates that the connected account is that role in the network. For example, if a shareholder of a company who is not the listed controller tries to access the *company controller* section of the prototype, they will be reject as shown in figure 9.7.


Smart SAFEs
localhost:3000/company-controller/0xDf007707210aD7F30251EA0072b85577f49f8CDc
Apps
Gmail
CBA Netbank
Messenger
Thesis Drive
COMP9517
Reading List

**NOTE:** You are not the controller! Please join the network as a different role or use a different company contract address.

Figure 9.7: Prototype rejecting actor who is not the listed controller of a company

Even if an attacker manages to bypass this part of the prototype and gain access to a screen that they should not be allowed to access, all of the permissioning is still handled by the network and the smart contracts. This means that even in this situation, when their transaction is signed and sent to the blockchain, it will be rejected by the smart contracts.

An actor does not need to use the prototype to interact with the smart contracts on the network - they can sign and send transactions manually with whatever tools they like. The prototype only acts as a convenience tool to interact with the network and as such, is not a point of vulnerability for an attacker to compromise the network.

## 9.4 Future Work

This section describes the limitations of the prototype and suggests some features that could be implemented to improve it.

### 9.4.1 Entity details

Currently the prototype only shows a little bit of information about the entities that are in the network. Entities like companies, SAFEs or investors only exist in tables with a few columns showing basic information like the name, the account balance, total number of SAFEs issues etc. For each of these entities, it would be beneficial to have a relevant screen that has more information. For example, for a regulator, it would be useful for them to click on a company in the *Companies* table and see more information about the founder, the company status, which regulator registered this company etc.

### 9.4.2 Company history

One of the benefits of using a blockchain is that all data is open and there is high assurance of its integrity. Regulators would benefit from being able to inspect the history of each company and view all actions that were performed on the company smart contract. It is very natural for a regulator to want to know the history of a company's equity rounds, any payments it has made to employees as well the SAFEs that it has issued and who they were issued to.

Building out something like this would require more analysis on the permissioning requirements of a regulator and ensuring that any information that a company wants to keep private can remain so.

### 9.4.3 Investor smart contract

Right now investors are just represented as Ethereum addresses on the network. This means properties such as the investor's name or whether they are a licensed accredited investor cannot be captured. An *Investor* smart contract could be written to capture all of this information. Similarly to how the *RolesManager* contract manages all of the registered companies, and a company looking to register on the network needs to do so through a regulator, the same behaviour could be encoded for an investor.

Once again, this feature would require further research into what information is important to a regulator and what can remain private to the investor.

### 9.4.4 View multiple companies at once

As mentioned in section 9.1, an investor or company controller can only view one company at a time. It is desirable that an actor who connects to the GUI is able to view all investments that they have across multiple companies on the network. This could be implemented by, once again, creating an Investor smart contract that maintains a list of all the companies that an investor is involved with. The address for this smart contract can then be supplied to the GUI to load information.

# Chapter 10

# Conclusion

This paper included a comprehensive background on blockchain and the SAFE equity-financing contract discussed in chapter 2. A motivation for this project was presented in chapter 3.This was followed by a literature review in chapter 4 which explored other types of decentralised equity-financing structures such as the ICO, and the challenges of representing law as code. A problem statement was then presented in chapter 5 which presented the problems of other projects in this paper's space and explained the goal of this paper. This was followed by further research and implementation of the smart contract designs (6), a permissioned private chain (8) and prototype and GUI (9).

Reflecting on section 5, the goal of *Building Smart SAFEs* was to further understand the challenges of implementing a SAFE contract using a blockchain, through the construction of a prototype. By the end of this project, it is clear that there are many challenges to address and creating a functional prototype that can be applied to real businesses will require more than one year of individual work.

The value added through this project was:

1. Furthering smart contracts designs that built on the existing work from *Architecture for SAFE Smart Contracts* [RvdM21a] to support Liquidity and Dissolution

Events of the SAFE contract.

2. An analysis of the permissioning requirements for a network that could be used for Australian startups funded with SAFE contracts. This involved using a Besu private chain and setting up an on-chain permissioning structure.
3. A prototype that encompassed a locally running private chain and GUI to be used for a regulator, company investor and company founder. This brought up some interesting questions of how a company can be on-boarded and how flexible permissions can be enforced.

Something unexpected that came out of this work was the application of game-theory between SAFE investors in a Liquidity Event, explained in chapter 7.

## 10.1 Future Work

This project was broad in the sense that it touched on multiple topics. Areas for future work have been noted throughout this document and since this project sits in the context of *Can SAFEs Be Smart?* [RvdM21b], it is natural for more work to be completed on this topic.

Below are a few areas of extension that can be implemented directly on top of this project and will progress the prototype towards a useable application:

- Create a view for a company's historical transactions. As discussed in section 9.4.2, all of this information is captured by the blockchain, it just needs to be fed through into the GUI. This is important since both regulators and company controllers would be interested in viewing a ledger of the company's history. See section 9.4 for more suggestions to the prototype.
- Allow users to design their own company smart contracts without compromising the network's permissioning requirements. This was discussed in section 8.2.4 and

is a valuable addition since company's vary significantly in structure so it would be effective to allow companies to specify their own smart contract structure. This could be achieved by creating a Company interface that company smart contracts have to conform to[1].

- Extend the Liquidity Event to support non-cash payments. This was discussed in section 6.1. This could be achieved by representing company shares a smart contract that follows the ERC-20 token standard for fungible tokens [cor19]. The *Liquidity_Event* smart contract code will need to be changed significantly to support this.

[1]Although since Solidity is not a statically typed language, enforcing a smart contract to follow this interface at runtime would require some creative thought.

# Bibliography

[And21] Somer Anderson. What is game theory. https://www.investopedia.com/terms/g/gametheory.asp#, 2021. Accessed: 2021-11-10.

[Ber19a] Blockchainhub Berlin. Dao. https://blockchainhub.net/wp-content/uploads/2019/07/DAOs_TaditionalOrganisations.png, 2019. Accessed: 2021-04-21.

[Ber19b] Blockchainhub Berlin. Traditional top down organisation. https://blockchainhub.net/wp-content/uploads/2019/07/DAOs_DecentralizedAutonomousOrganizations.png, 2019. Accessed: 2021-04-21.

[Bes20] Hyperledger Besu. Permissioning. https://besu.hyperledger.org/en/stable/Concepts/Permissioning/Permissioning-Overview/#permissioning, 2020. Accessed: 2021-11-17.

[Bes21a] Hyperledger Besu. Allowlist. https://besu.hyperledger.org/en/stable/HowTo/Limit-Access/Local-Permissioning/#node-allowlisting, 2021. Accessed: 2021-11-17.

[Bes21b] Hyperledger Besu. Allowlist. https://besu.hyperledger.org/en/stable/HowTo/Limit-Access/Local-Permissioning/#updating-the-node-allowlist, 2021. Accessed: 2021-11-17.

[Bes21c] Hyperledger Besu. Ibft 2.0. https://besu.hyperledger.org/en/stable/HowTo/Configure/Consensus-Protocols/IBFT/#ibft-20, 2021. Accessed: 2021-11-17.

[Bry] Sean Bryant. How many startups fail and why? https://www.investopedia.com/articles/personal-finance/040915/how-many-startups-fail-and-why.asp. Accessed: 2021-04-21.

[Con] Bill Conerly. High frequency trading explained simply. https://www.forbes.com/sites/billconerly/2014/04/14/high-frequency-trading-explained-simply/?sh=449b2c123da8. Accessed: 2021-04-21.

[cor19] corwintines. Erc-20 token standard. https://ethereum.org/en/developers/docs/standards/tokens/erc-20/, 2019. Accessed: 2021-11-17.

[ESwn] Ekaterina SHKARBUTA Elena SELIVANOVA. What are the fundamental problems in the u.s. ico market? https://sbh-partners.com/en/what-are-the-fundamental-problems-in-the-us-ico-market#, Unknown. Accessed: 2021-04-27.

[Eth] Ethereum. Ethereum. https://www.linkedin.com/company/ethereum/. Accessed: 2021-04-21.

[Fra20] Jake Frankenfield. Consensus mechanism (cryptocurrency). https://www.investopedia.com/terms/c/consensus-mechanism-cryptocurrency.asp, 2020.

[Fra21] Jake Frankenfield. Smart contracts. https://www.investopedia.com/terms/s/smart-contracts.asp, 2021. Accessed: 2021-04-27.

[Geo] Geora. https://www.geora.io/. Accessed: 2021-04-21.

[Gra] Paul Graham. Announcing the safe, a replacement for convertible notes. https://ycombinator.wpengine.com/announcing-the-safe-a-replacement-for-convertible-notes/. Accessed: 2021-04-21.

[Gro18] Satis Research Group. Cryptoasset market coverage initiation: Network creation. https://research.bloomberg.com/pub/res/d28giW28tf6G7T_Wr77aU0gDgFQ, 2018. Accessed: 2021-04-27.

[Hay21] Adam Hayes. Liquidity event. https://www.investopedia.com/terms/l/liquidity_event.asp#, 2021. Accessed: 2021-06-27.

[Kha21] Khadija Khartit. Capital stock. https://www.investopedia.com/terms/c/capitalstock.asp#, 2021. Accessed: 2021-11-10.

[Lee17] Charlie Lee. Did a cross-chain atomic swap. https://twitter.com/satoshilite/status/911328252928643072?lang=en, 2017. Accessed: 2021-04-26.

[Lev18] Carolynn Levy. Safe financing documents. https://www.ycombinator.com/documents/, September 2018. Accessed: 2021-04-21.

[LL18] Stuart D. Levi and Alex B. Lipton. An introduction to smart contracts and their potential and inherent limitations. https://corpgov.law.harvard.edu/2018/05/26/an-introduction-to-smart-contracts-and-their-potential-and-inherent-limitations/, May 2018.

[Mar19a] Andrew Marshall. Cointelegraph. https://cointelegraph.com/explained/ico-explained, 2019. Section 1.

[Mar19b] Andrew Marshall. Cointelegraph. https://cointelegraph.com/explained/ico-explained, 2019. Section 3.

[Plu] Plus500. What are the most traded cryptocurrencies. https://www.plus500.com.au/Trading/CryptoCurrencies/What-are-the-Most-Traded-Cryptocurrencies~2#. Accessed: 2021-04-27.

[PwC18] PwC. Pwc's global blockchain survey. https://www.pwc.com/gx/en/industries/technology/blockchain/blockchain-in-business.html, 2018. Accessed: 2021-04-27.

[PwC19] PwC. Cryptographic assets and related transactions: accounting considerations under ifrs. https://www.pwc.com/gx/en/audit-services/ifrs/publications/ifrs-16/cryptographic-assets-related-transactions -accounting-considerations-ifrs-pwc-in-depth.pdf, 2019. Page 2.

[Ral] Geoff Ralston. A guide to seed fundraising. https://www.ycombinator.com/library/4A-a-guide-to-seed-fundraising. Accessed: 2021-04-27.

[Rei20] Nathan Reiff. Blockchain technology's three generations. https://www.investopedia.com/tech/blockchain-technologys-three-generations/, 2020. Accessed: 2021-04-27.

[Rub94] Martin J Osborne; Ariel Rubinstein. *A Course in Game Theory.* Cambridge, 1994. ISBN 9780262150415, page 14.

[RvdM21a] Michael Maher Ron van der Meyden. Architecture for safe smart contracts. Manuscript not published, 2021.

[RvdM21b] Michael Maher Ron van der Meyden. Can safes be smart? Manuscript not published, 2021.

[SHA19] TOSHENDRA KUMAR SHARMA. Public vs. private blockchain : A comprehensive comparison. https://www.blockchain-council.org/blockchain/public-vs-private-blockchain-a-comprehensive-comparison/, 2019. Accessed: 2021-04-27.

[She19] Benjamin Sherry. What is an ico? https://www.investopedia.com/news/what-ico/, 2019. Accessed: 2021-04-27.

[SHW19] Roberto Saltini and David Hyland-Wood. Ibft 2.0: A safe and live variation of the ibft blockchain consensus protocol for eventually synchronous networks, 09 2019.

[Sie16] David Siegel. Understanding the dao attack. https://www.coindesk.com/understanding-dao-hack-journalists, June 2016.

[SSD18] SSD. A deep dive on end-to-end encryption. https://ssd.eff.org/en/module/deep-dive-end-end-encryption-how-do-public-key-encryption-systems-work, 2018. Accessed: 2021-04-26.

[Tac21] Caner Tacoglu. Trustless. https://academy.binance.com/en/glossary/trustless, 2021. Accessed: 2021-04-26.

[Tex] Open Text. Differentiate between centralized and decentralized management. https://opentextbc.ca/principlesofaccountingv2openstax/chapter/differentiate-between-centralized-and-decentralized-management/. Accessed: 2021-04-26.

[vdM21] Ron van der Meyden. A game theoretic analysis of liquidity events in convertible instruments. Manuscript not published, 2021.

[Vos19] Shermin Voshmgir. Tokenized networks: What is a dao? https://blockchainhub.net/dao-decentralized-autonomous-organization/, July 2019.

[YCo21] YCombinator. Simple agreement for future equity. https://www.ycombinator.com/assets/ycdc/PostmoneySafe-ValuationCapOnlyv1.1-cb4934724e32a5864fd43ca894e51c8c081ecef136409c2e94f00d92168230ec.docx, 2021. Accessed: 2021-11-10.

[Zegwn] Zegal. Overview of a safe pro rata rights agreement. https://zegal.com/en-au/safe-pro-rata-rights-agreement/, Unknown. Accessed: 2021-04-27.

# Appendix 1

## A.1 Code for Smart Contracts

All contracts directory: `https://github.com/william-coulter/Thesis/blob/master/src/contracts/Company.sol`

Company: `https://github.com/william-coulter/Thesis/blob/master/src/contracts/Company.sol`

Safe_Controller: `https://github.com/william-coulter/Thesis/blob/master/src/contracts/SAFEs/SafeController.sol`

Liquidity_Event: `https://github.com/william-coulter/Thesis/blob/master/src/contracts/Events/LiquidityEvent.sol`

Dissolution_Event: `https://github.com/william-coulter/Thesis/blob/master/src/contracts/Events/DissolutionEvent.sol`

Roles_Manager: `https://github.com/william-coulter/Thesis/blob/master/src/contracts/Permissioning/RolesManager.sol`

## A.2 Genesis file for IBFT 2.0 private chain

The file is too wide to paste in this document.

`https://github.com/william-coulter/Thesis/blob/master/src/network/genesis.json`

## A.3 Liquidity Definition

"Change of Control" means (i) a transaction or series of related transactions in which any "person" or "group" (within the meaning of Section 13(d) and 14(d) of the Securities Exchange Act of 1934, as amended), becomes the "beneficial owner" (as defined in Rule 13d-3 under the Securities Exchange Act of 1934, as amended), directly or indirectly, of more than 50% of the outstanding voting securities of the Company having the right to vote for the election of members of the Company's board of directors, (ii) any reorganization, merger or consolidation of the Company, other than a transaction or series of related transactions in which the holders of the voting securities of the Company outstanding immediately prior to such transaction or series of related transactions retain, immediately after such transaction or series of related transactions, at least a majority of the total voting power represented by the outstanding voting securities of the Company or such other surviving or resulting entity or (iii) a sale, lease or other disposition of all or substantially all of the assets of the Company.

## A.4 Dissolution Definition

"Dissolution Event" means (i) a voluntary termination of operations, (ii) a general assignment for the benefit of the Company's creditors or (iii) any other liquidation, dissolution or winding up of the Company (excluding a Liquidity Event), whether voluntary or involuntary.

## A.5 Prototype Screenshots

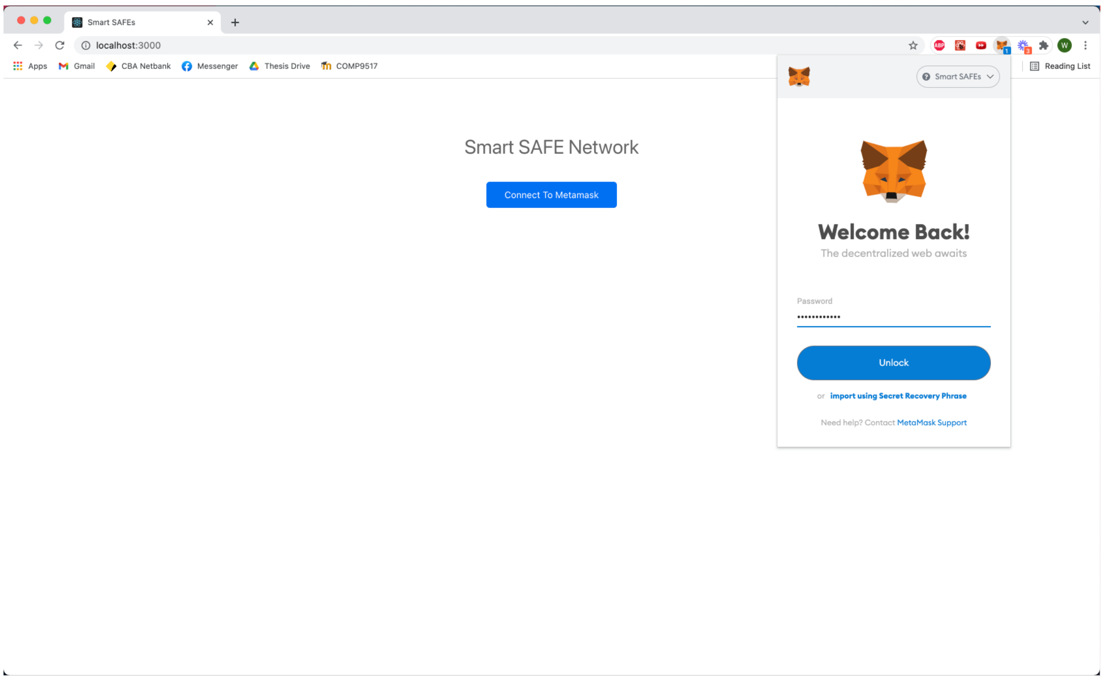
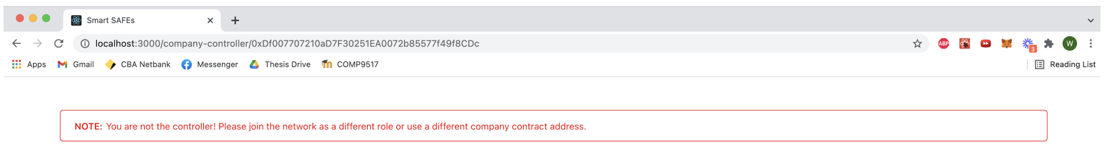
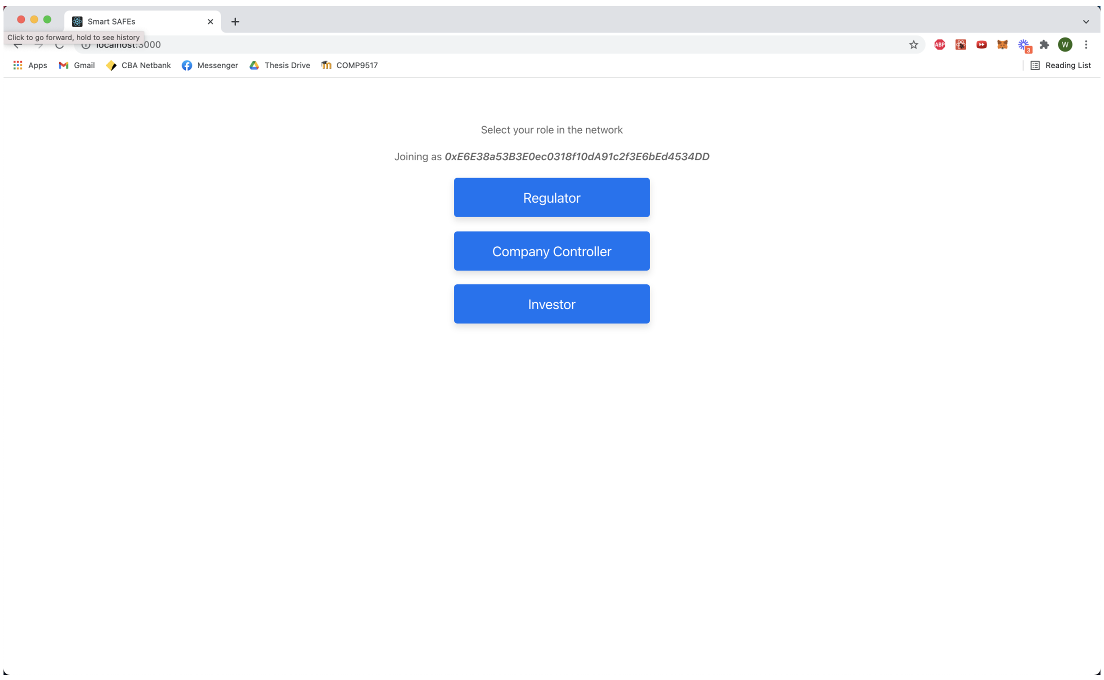

Smart SAFEs
localhost:3000
Smart SAFE Network
Connect To Metamask
Welcome Back!
The decentralized web awaits
Password
Unlock
or import using Secret Recovery Phrase
Need help? Contact MetaMask Support

Figure A.1: Prototype: Connect to Metamask


Smart SAFEs
localhost:3000/company-controller/0xDf007707210aD7F30251EA0072b85577f49f8CDc

**NOTE:** You are not the controller! Please join the network as a different role or use a different company contract address.

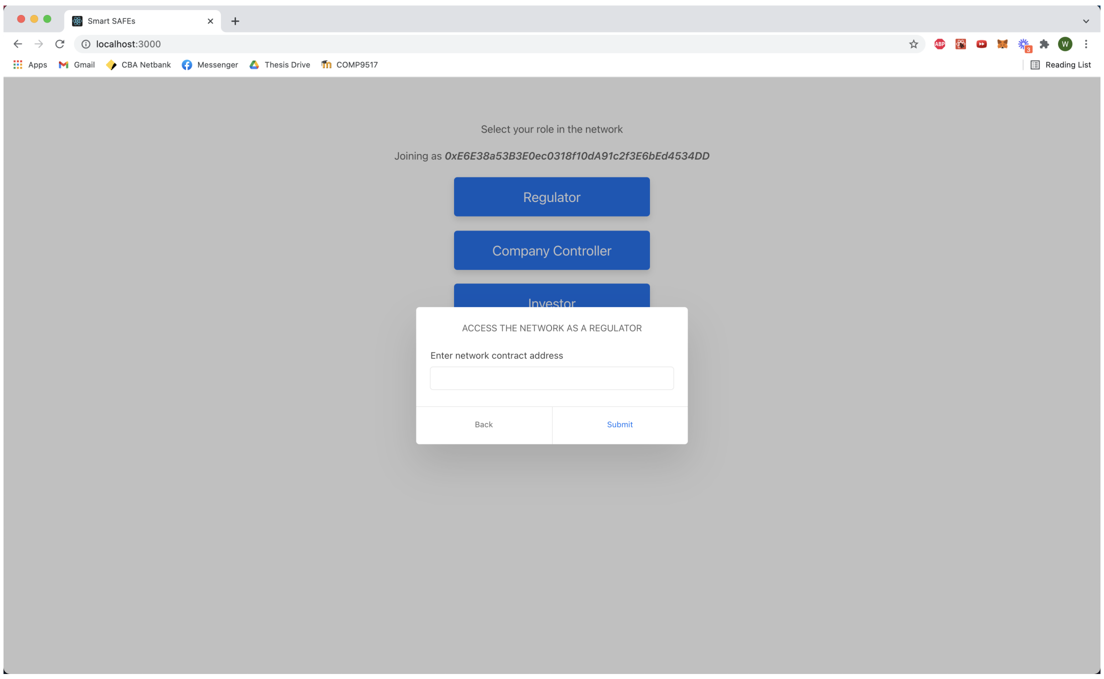
Figure A.2: Prototype: No permission to view screen



Select your role in the network

Joining as *0xE6E38a53B3E0ec0318f10dA91c2f3E6bEd4534DD*

Figure A.3: Prototype: Select Role



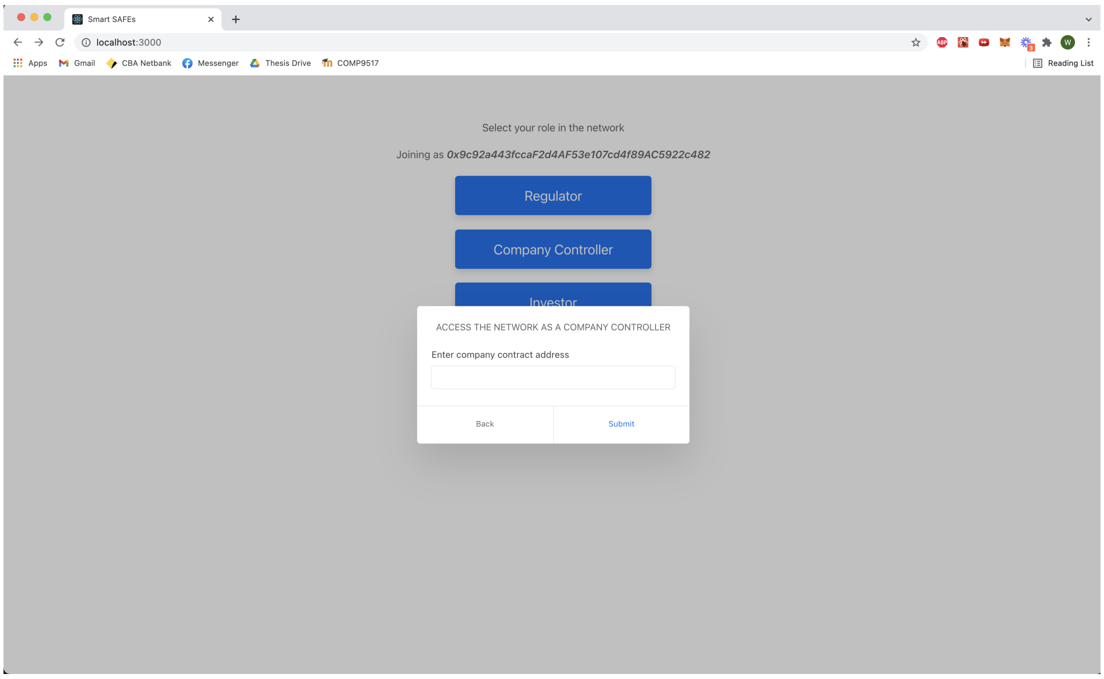
Select your role in the network

Joining as *0xE6E38a53B3E0ec0318f10dA91c2f3E6bEd4534DD*

ACCESS THE NETWORK AS A REGULATOR

Enter network contract address

Figure A.4: Prototype: Join as regulator



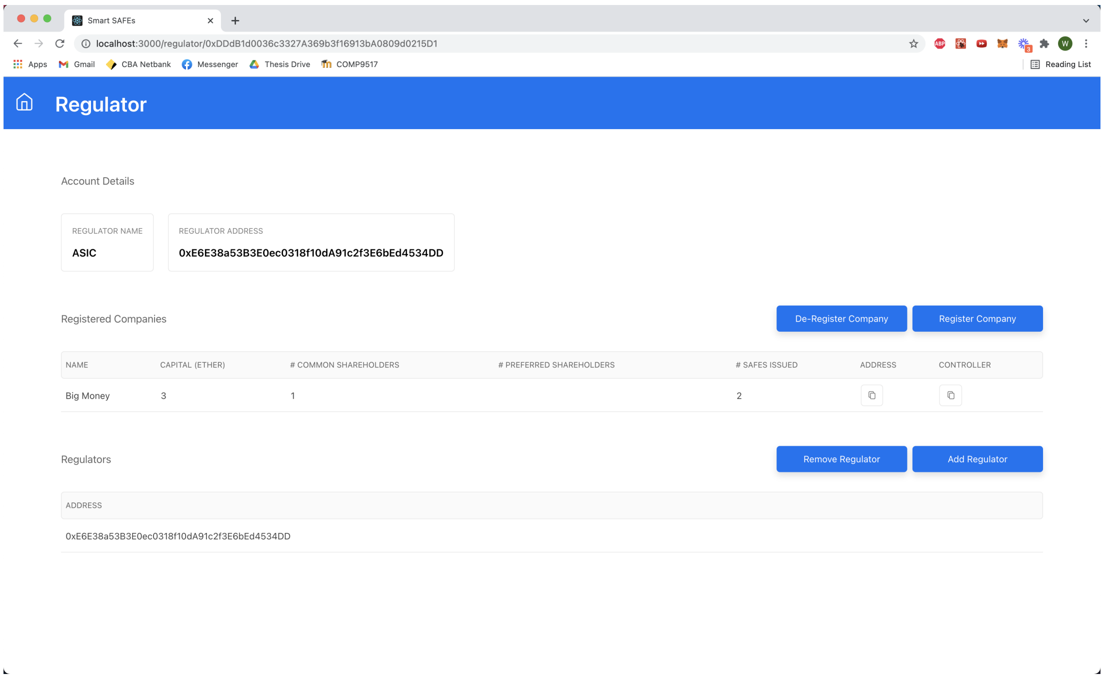
Select your role in the network

Joining as *0x9c92a443fccaF2d4AF53e107cd4f89AC5922c482*

ACCESS THE NETWORK AS A COMPANY CONTROLLER

Enter company contract address

Figure A.5: Prototype: Join as controller



Account Details

| REGULATOR NAME | REGULATOR ADDRESS |
|---|---|
| ASIC | 0xE6E38a53B3E0ec0318f10dA91c2f3E6bEd4534DD |

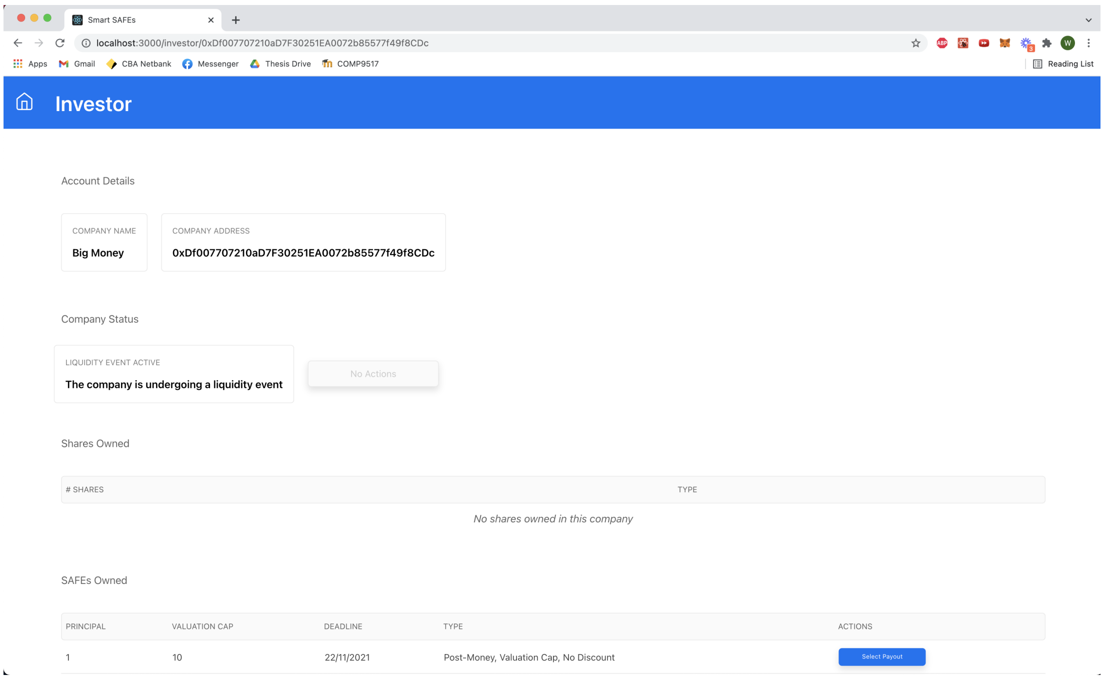
Registered Companies

| NAME | CAPITAL (ETHER) | # COMMON SHAREHOLDERS | # PREFERRED SHAREHOLDERS | # SAFES ISSUED | ADDRESS | CONTROLLER |
|---|---|---|---|---|---|---|
| Big Money | 3 | 1 | | 2 | | |

Regulators

| ADDRESS |
|---|
| 0xE6E38a53B3E0ec0318f10dA91c2f3E6bEd4534DD |

Figure A.6: Prototype: Regulator Screen



Investor

Account Details

| COMPANY NAME | COMPANY ADDRESS |
|---|---|
| Big Money | 0xDf007707210aD7F30251EA0072b85577f49f8CDc |

Company Status

LIQUIDITY EVENT ACTIVE
The company is undergoing a liquidity event

Shares Owned

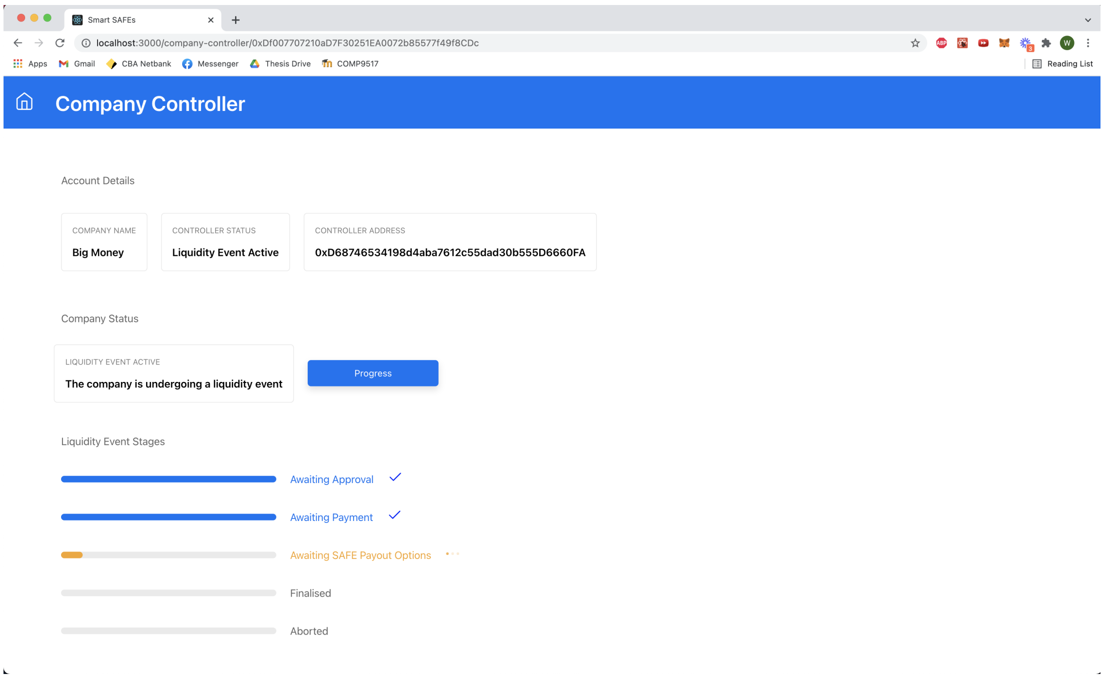
| # SHARES | TYPE |
|---|---|

*No shares owned in this company*

SAFEs Owned

| PRINCIPAL | VALUATION CAP | DEADLINE | TYPE | ACTIONS |
|---|---|---|---|---|
| 1 | 10 | 22/11/2021 | Post-Money, Valuation Cap, No Discount | Select Payout |

Figure A.7: Prototype: Investor Screen


Smart SAFEs
localhost:3000/company-controller/0xDf007707210aD7F30251EA0072b85577f49f8CDc
Apps
Gmail
CBA Netbank
Messenger
Thesis Drive
COMP9517
Reading List

Company Controller

Account Details

| COMPANY NAME | CONTROLLER STATUS | CONTROLLER ADDRESS |
|---|---|---|
| Big Money | Liquidity Event Active | 0xD68746534198d4aba7612c55dad30b555D6660FA |

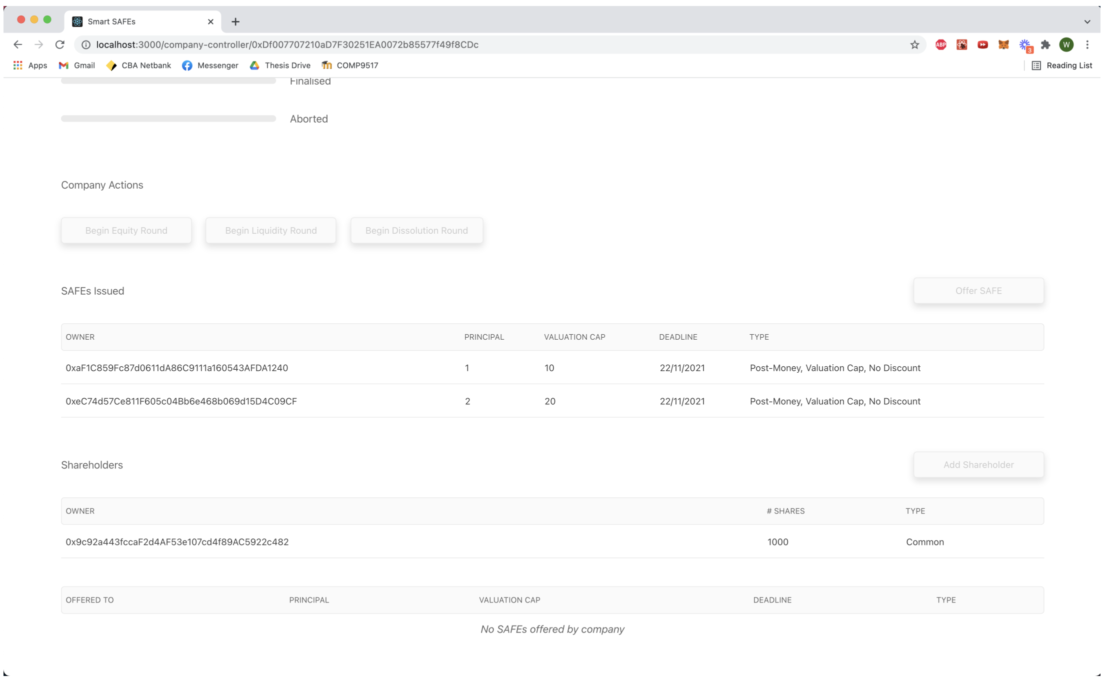
Company Status

LIQUIDITY EVENT ACTIVE
The company is undergoing a liquidity event

Liquidity Event Stages

Awaiting Approval ✓
Awaiting Payment ✓
Awaiting SAFE Payout Options
Finalised
Aborted

Figure A.8: Prototype: Controller Screen 1


localhost:3000/company-controller/0xDf007707210aD7F30251EA0072b85577f49f8CDc

Finalised

Aborted

Company Actions

SAFEs Issued

| OWNER | PRINCIPAL | VALUATION CAP | DEADLINE | TYPE |
|---|---|---|---|---|
| 0xaF1C859Fc87d0611dA86C9111a160543AFDA1240 | 1 | 10 | 22/11/2021 | Post-Money, Valuation Cap, No Discount |
| 0xeC74d57Ce811F605c04Bb6e468b069d15D4C09CF | 2 | 20 | 22/11/2021 | Post-Money, Valuation Cap, No Discount |

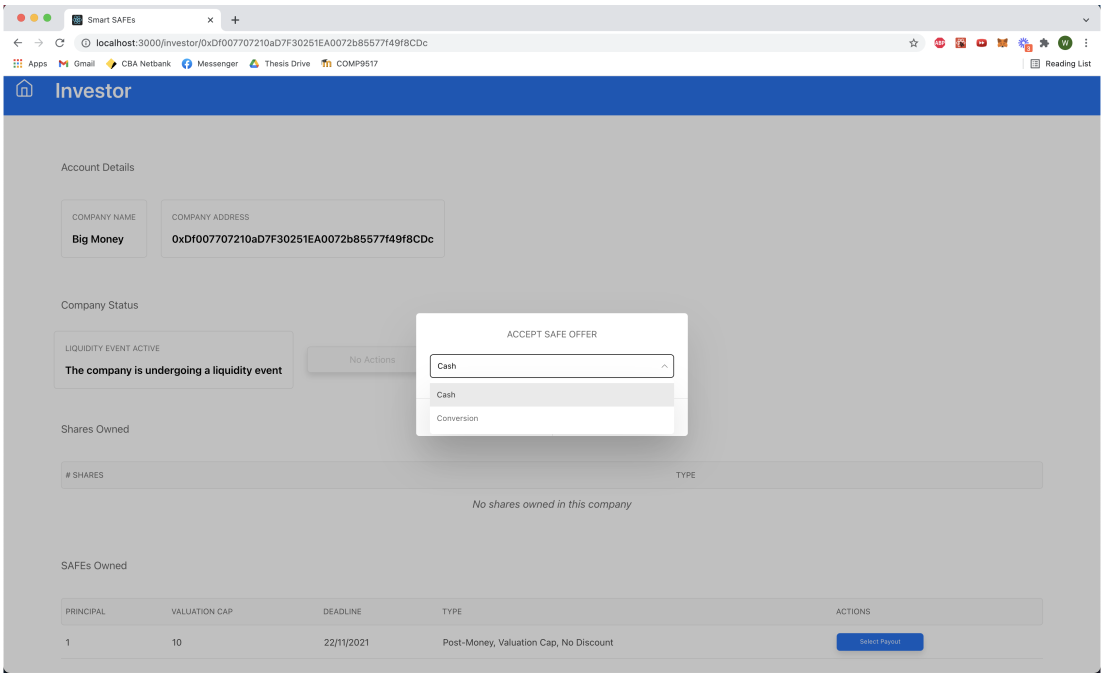
Shareholders

| OWNER | # SHARES | TYPE |
|---|---|---|
| 0x9c92a443fccaF2d4AF53e107cd4f89AC5922c482 | 1000 | Common |

| OFFERED TO | PRINCIPAL | VALUATION CAP | DEADLINE | TYPE |
|---|---|---|---|---|

*No SAFEs offered by company*

Figure A.9: Prototype: Controller Screen 2


localhost:3000/investor/0xDf007707210aD7F30251EA0072b85577f49f8CDc

Investor

Account Details

COMPANY NAME: **Big Money**

COMPANY ADDRESS: **0xDf007707210aD7F30251EA0072b85577f49f8CDc**

Company Status

LIQUIDITY EVENT ACTIVE: **The company is undergoing a liquidity event**

Shares Owned

| # SHARES | TYPE |
|---|---|

*No shares owned in this company*

SAFEs Owned

| PRINCIPAL | VALUATION CAP | DEADLINE | TYPE | ACTIONS |
|---|---|---|---|---|
| 1 | 10 | 22/11/2021 | Post-Money, Valuation Cap, No Discount | Select Payout |

Figure A.10: Prototype: SAFE Investor select payout