

THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

COLLEGE OF INFORMATION SCIENCES AND TECHNOLOGY

APPLYING NAKAMOTO: ENABLING ALTERNATIVE METHODS OF FUNDING FOR
INDEPENDENT ONLINE CONTENT CREATORS

PHILIP CHWISTEK
SPRING 2019

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Information Sciences and Technology and English
with honors in Information Sciences and Technology

Reviewed and approved* by the following:

David Reitter
Associate Professor of Information Sciences and Technology
Thesis Supervisor

Steven Haynes
Teaching Professor of Information Sciences and Technology
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

This paper concerns itself with the question, “how can we apply blockchain technology to develop an alternative subscription model for independent online creators?” The proposed solution, Submerged, is a hybrid application that is a combination of traditional web technologies and smart contracts, in the form of what is referred to in the blockchain space as a *decentralized application (dapp)*. Following EOSIO dapp general practices, Submerged integrates with ScatterJS: a wallet software designed to specifically interact with dapps. Following the paradigm described by Hevner as the “three-cycle view” of design research, this paper outlines the problems and opportunities in the crowdfunding space, proposes an alternative form of interaction between audiences and creators, provides an overview of an implemented minimally-viable-product, and performs an evaluation to determine how Submerged improves on what already exists in the crowdfunding domain. Having implemented and tested Submerged in a staging environment, this paper concludes with a section detailing how the current iteration can be improved.

TABLE OF CONTENTS

LIST OF FIGURES	iii
LIST OF TABLES	iv
ACKNOWLEDGEMENTS	v
Chapter 1 Introduction	1
Purpose.....	1
Chapter 2 Literature Review	4
What is Money?	4
Blockchain	5
Bitcoin Protocol	6
Wallets and Accounts.....	8
Smart Contracts, Tokens, and Ethereum	9
EOSIO	11
Existing Decentralized Applications	14
Chapter 3 Methodology	17
Chapter 4 Submerged – Relevance Cycle.....	18
The Problem Domain	18
Technological Constraints/Dependencies	19
Chapter 5 Submerged – Design Cycle	21
Choosing a Blockchain	21
A New, Formalized Relationship Between Creators and Audiences	22
The SUBM Token	26
The Submerged Foundation	26
The Submerged Smart Contract	27
Who Pays? Users or the Developers?	33
The Submerged Application	34
The Submerged Application Walkthrough	36
Chapter 6 Submerged – Rigor Cycle	47
Methodology for Estimating Contract Costs.....	47
Estimating User Costs	47
Estimating Deployment Costs.....	50

Depending on EOSIO	52
Issues Regarding User Experience.....	52
Security & Privacy	53
Chapter 7 Future Work	55
EOS Account System.....	55
Stable Coins	55
A Potential (Obstructed) Path to Launch	56
Chapter 8 Final Remarks	58
Submerged in Review	58
Implications.....	58
Appendix A Links to GitHub Repo and Relevant Documentation/Tools	61
Submerged Repository	61
EOSIO Repository	61
Scatter Documentation	61
NestJS Documentation	62
ReactJS Documentation	62
Redux Documentation.....	62
Appendix B Learning Resources	63
EOSIO.....	63
JavaScript Development.....	63
BIBLIOGRAPHY.....	65

LIST OF FIGURES

Figure 1. Gartner's Hype Cycle.....	2
Figure 2 A Merkle Tree	6
Figure 3 Diagram of Submerged Model	25
Figure 4 Contract Pseudocode	30
Figure 5 Submerged UML Sequence Diagram.....	35
Figure 6 Creating a Submerged Account.....	36
Figure 7 Linking Scatter to Submerged	37
Figure 8 Login into Submerged	37
Figure 9 Scatter Identity.....	37
Figure 10 Creating a Channel	38
Figure 11 Channel Summary	38
Figure 12 Scatter Modal.....	39
Figure 13 Scatter Create Channel Confirmation.....	39
Figure 14 Channel listed in Channels List.....	40
Figure 15 A User's Channel View.....	40
Figure 16 Declaring a Project	41
Figure 17 Feed Populated with Declared Projects	42
Figure 18 Scatter Subscription Confirmation	42
Figure 19 Dashboard View Post Subscription	43
Figure 20 Feed Event For Delivered Project	43
Figure 21 Scatter Vote Confirmation.....	44
Figure 22 Channel Summary	45
Figure 23 Project Detail Modal with Incomplete Project	46
Figure 24 Project Detail Model with Delivered Project.....	46

Figure 25 Submerged v. Patreon Equation50

LIST OF TABLES

Table 1 Submerged Terminology	22
Table 2 Multi-Index Tables in the Submerged Contract.....	29
Table 3 Smart Contract Actions.....	32
Table 4 EOS Allocation For Users/Creators.....	48
Table 5 EOS Resource Allocation for Deployment with 50,000 users.....	50

ACKNOWLEDGEMENTS

I would like to thank both Dr. David Reitter and Dr. Steven Haynes for not only advising me through the thesis process but for also offering me numerous opportunities to work with them independently during my undergraduate career. Without their guidance and wisdom, I would not be in the same place where I am today. I would also like to thank my family, who have encouraged me wholeheartedly through my education and continue to offer me their undying support.

Chapter 1

Introduction

Purpose

For most people, blockchain is synonymous with Bitcoin and other cryptocurrencies such as Ether, Ripple, and Litecoin, or in other words, the speculative bubble that took place in late 2017. But beneath the headlines lies a technology that has substantial potential for reducing the presence of intermediaries and facilitating the management of digital assets including identity, voting, art, and of course, digital money. Despite attracting the likes of IBM, JP Morgan Chase, and Mastercard, blockchain is still underdeveloped and has few practical, existing applications. Startups promising blockchain-based electronic medical records, peer-to-peer energy grid transactions, and decentralized corporate structures are frequently far-fetched, ignoring the technical and legislative limitations that exist. The purpose of this research project is to investigate the scalability and usability of using a blockchain infrastructure to support subscriptions, escrow, and fund distribution. Acknowledging the realities of the blockchain in its current form should help pave the way for more advanced applications such as those enumerated above.

At the time that this paper was written, general enthusiasm for blockchain-enabled technologies and applications has been in decline, following the rapid collapse of the market valuation for cryptocurrencies. Therefore, the current period can be identified as the “Trough of Disillusionment” for blockchain-related technology according to Gartner’s Hype Cycle (see

figure 1) [1]. The “Peak of Inflated Expectations” parallels the peak of cryptocurrency valuations in late 2017, where the price of Bitcoin hovered around \$20,000 [2]. Ultimately, the end-goal of this research is to provide a stepping stone in the path up the “Slope of Enlightenment.” To provide a valuable example of a decentralized application, it is important to pursue an earnest solution: not simply make a to-do list on the blockchain. For this reason, this project builds around the question “how can we apply blockchain technology to develop an alternative subscription model for independent online creators?”

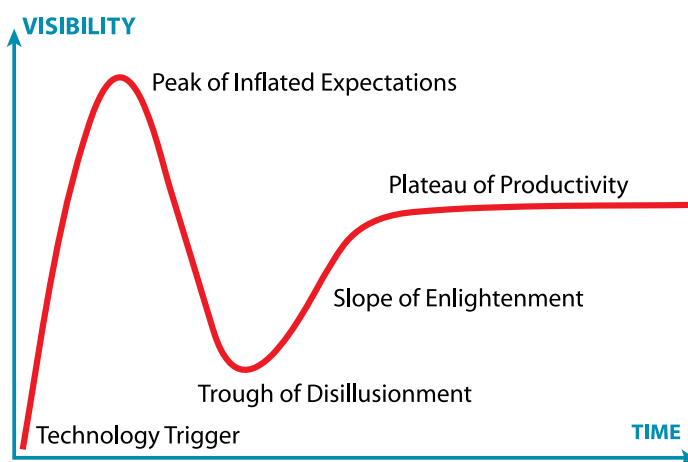


Figure 1. Gartner's Hype Cycle [3]

Internet content creators have long struggled to monetize their work. Prior to the advent of Patreon, content creators, especially YouTubers and podcasters, were limited to raising funds through advertisers or through more informal donation avenues (PayPal). Since Patreon’s inception in 2013, the crowdfunding space has expanded to include other fundraising methods such as YouTube Channel Subscriptions. Nonetheless, these services are largely based on the goodwill between audiences and creators. Submerged, a response to the research question above, is a decentralized application that aims to improve on the existing crowdfunding domain by reducing the fees paid by content creators, enforcing a creator’s self-determined production

schedule to provide subscribers with a guarantee of content, and compensating creators proportionally to the traffic they bring to the platform.

Chapter 2

Literature Review

What is Money?

The answer seems obvious. We use it to buy things such as groceries, clothes, and books. We use it to compensate the dogsitter. We use it to pay for our Netflix subscriptions. Even a toddler running a play store knows that goods are exchanged for money. So why does the definition of “money” even matter in the context of this paper?

Blockchain, the technology central to this project and foundational to cryptocurrencies like Bitcoin, is ultimately supported and maintained by a system of financial incentives. To understand blockchain, one must understand these incentives, which in turn require a deeper understanding of money. When individuals are first exposed to the concept of a cryptocurrency, one of their first inclinations may be to dismiss the concept as a fad and declare that cryptocurrencies have “no intrinsic value.” This statement is indeed true, but paper money also has no intrinsic value, yet we can use it at any retail location.

At the beginning of the turn of the twentieth century, policymakers around the world debated over what the best monetary system would look like: gold, silver, bimetallism, et cetera. George Friedrich Knapp, a German economist, published a seminal work that argued money can and should exist without a metallic standard [4]. He postulated that any item could be used as a currency as long as it gathered enough social consensus and had qualities that deemed it an effective “item of exchange” and more precisely, an effective “means of payment.” The first money-wielding humans used gold to fulfill transactions not because of its industrial worth (it has little), but because it satisfied the requirements to be an effective item of exchange. Put

simply, precious metals are scarce, durable, and can be easily confirmed as authentic. Paper money, too, with the help of a central bank, maintains scarcity, is produced in a manner that is difficult to counterfeit, and can be verified as genuine. Therefore, our use of the United States dollar is based on our trust in American institutions, not the amount of gold in Fort Knox that the bill supposedly represents. If anything, the precious metals maintained by the US Treasury are there as backup “exchange commodities” in case the dollar ever fails.

One can imagine a blockchain as a technological method of ensuring a digital asset’s scarcity and verifiable authenticity – thus opening up the potential for such an asset to be considered an effective “item of exchange.” Understanding social consensus as the most important aspect of whether an item has value is key to understanding blockchain and, in a broader sense, how decentralized applications, like the one outlined in this paper, work.

Blockchain

The first mention of a “block chain” appeared in a series of papers by Haber and Stornetta in the early ’90s as a method to timestamp documents and ensure their integrity [5]. In the original version of this proposed architecture, a server would receive a document from a client and then create a digital certificate comprised of the previous document’s certificate (a hash pointer) and the current time, thus creating a chain of documents. Any change to any of the documents, retrospectively, would invalidate the chain. If copies of this document chain existed across multiple clients, these clients could compare their certificates to establish a shared history. Later, Haber and Stornetta improved on this schema to make it more computationally efficient to verify documents. Instead of linking these timestamped documents linearly, it is more effective

to group documents into “blocks,” thus creating a “block chain,” where each block makes use of a data structure known as a *Merkle Tree* (see figure 2) [6] [7]. In the case of Bitcoin, and other Bitcoin-like cryptocurrencies, each hash is representative of a transaction.

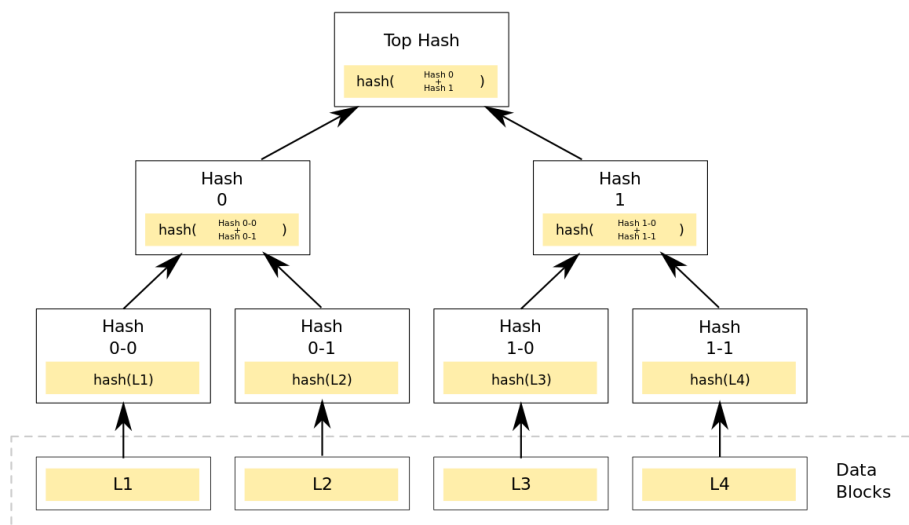


Figure 2 A Merkle Tree [8]

Bitcoin Protocol

Bitcoin, designed and implemented by an anonymous individual known by the name of “Satoshi Nakamoto,” adapted the concepts of a blockchain from Haber and Stornetta to create a global ledger system that supports digital money. Rather than timestamp documents, this blockchain, which most people associate with the term, manages account balances and transactions. Furthermore, there is no central location of this “blockchain.” Instead, thousands of computers called *nodes* have their own local copy. These nodes work together to maintain their copies and ensure consistency across the network. As a result, all transactions and account balances are public. This redundancy ensures that when a user desires to spend x amount of

Bitcoin, the network can agree that this individual indeed has x amount of Bitcoin to spend. This verification is key to solving the so-called “double-spending” problem: it prevents people from spending Bitcoin they do not have [6].

One of Nakamoto’s most significant contributions revolves around *mining nodes* and *consensus algorithms*. Mining nodes, also referred to as *miners*, are responsible for validating transactions and placing them into blocks. Once a transaction is placed within a block and accepted by the network, the transaction is “executed.” For this effort, miners receive a reward in the form of newly-minted Bitcoin: an inflationary pressure and incentive to maintain the network [7]. Each Bitcoin can be traced back to its original block, ensuring its authenticity.

To prevent too many Bitcoin from being minted and causing runaway inflation, Nakamoto implemented a consensus algorithm called proof-of-work (POW). In order to delay the production of new blocks, each miner must complete a puzzle via the hashcash protocol, which was originally developed by Adam Black in 1997 as a proposal to limit email spam [9]. The puzzle involves discovering a value that, when hashed with the transactions in the block, produces a satisfactory number of leading zero bits. Producing this value, called a *nonce*, can only be done through trial-and-error and takes a significant amount of time. The difficulty of the puzzle increases as more miners and more computational power enter the network [10]. Therefore, miners are in competition, computationally, with one another to create a new block and generate a nonce value that satisfies the difficulty deemed by the protocol. Once a satisfactory nonce is generated, the network accepts this new block and rewards the miner. Besides issues of scalability, proof-of-work consensus algorithms have faced increasing criticism for their waste of resources. If a miner does not succeed in being the first to generate an appropriate nonce, then all the computation that they have gone through is effectively wasted.

PWC estimates that the Bitcoin network, alone, consumes roughly the same amount of energy in a year as the whole country of Ireland [11].

Transactions are also not necessarily executed in chronological order. Whenever a user sends a transaction, they pay a transaction fee to incentivize a miner to place their transaction within a block. The higher the transaction fee, then the faster the transaction will be executed. Currently, the average transaction fee for Bitcoin is between 25 and 50 cents. During times of peak traffic, such as during a panic sell-off in 2017, transaction fees jumped to \$55 dollars [12].

Wallets and Accounts

When someone “owns” a Bitcoin, they do not own it in the way that they might own a piece of jewelry. More precisely, when someone “owns” a Bitcoin, they own the rights to send that Bitcoin to someone else. All Bitcoins are tethered to the network and are associated with an address (an account). This address is generated by a *cryptocurrency wallet*. Each address is secured with public key cryptography. User addresses almost always correspond to a public key that is used to receive funds, while the private key is needed to sign transactions from that address [10]. Whenever someone sends Bitcoin, they use their private key to sign the transaction to produce a unique signature that can only be created using the public and private key. A transaction will be rejected by the network if it does not produce the correct signature. Therefore, it is of utmost importance to keep a private key secure. Anyone who has access to an account’s private key has access to use the funds associated with the account. A wallet is a piece of software that manages these private and public keys for the user. A *hot* wallet refers to a wallet that is connected to the internet; a *cold* wallet is not [13].

Smart Contracts, Tokens, and Ethereum

Other than Bitcoin, the most significant blockchain, architecturally speaking, is Ethereum. Specifically, Ethereum introduced support for *smart contracts*. Originally theorized by Nick Szabo, a smart contract “is a set of promises, specified in digital form, including protocols within which the parties perform on these promises” [14]. However, this term is rather misleading in our context because not all smart contracts on Ethereum perform the functions described by Szabo. In reality, a smart contract is a piece of code appended to the blockchain. When an application makes use of one or more smart contracts it is called a *decentralized application*. Smart contracts also have their own addresses and, like transactions, are public. This means that they can receive funds and can also be examined by other parties.

Smart contracts are useful for when two or more parties depend on some action to be performed by a third party in a transparent manner. A classic example is gambling [13]. A smart contract could be responsible for generating odds for a dice game, taking bets, and then distributing funds accordingly. First, because the smart contract is public, users can examine the code to make sure that odds are being fairly generated. Second, all these computations and transactions are handled by miners who are separated from the actual event. Their job is to only run the code in the contract. Therefore, once each party commits and sends funds to the contract, they cannot withhold money from the winner or refuse to participate. Events will precipitate the way they are described in the contract.

One of the key problems facing smart contracts is described as the “oracle problem” [15]. A smart contract can execute a transaction given some kind of input, but how does one make sure that this input is correct? In our previous gambling example, the smart contract was self-contained. But how would this smart contract work if parties were betting on an event exterior of

the blockchain, such as a boxing match? There are services that describe themselves as “oracles” that execute smart contract events based on API calls, but these “oracles” rely on trust. Augur, a decentralized prediction market on Ethereum, has set some groundwork by creating a democratic solution based that allows users to “vote” on the truth [16].

Ethereum, which is self-described as a “programmable” blockchain, supports an isolated runtime environment for smart contracts called the Ethereum Virtual Machine (EVM). Ethereum smart contracts need to be compiled from a higher-level language, such as Solidity, into EVM bytecode, where they are then uploaded to the Ethereum blockchain through an Ethereum client (usually a specialized wallet) [17]. Ethereum’s native cryptocurrency is Ether, which acts as a what Georg Friedrich Knapp would qualify as an “exchange commodity.” It can be used in the same way as Bitcoin or any other cryptocurrency but is primarily intended to be a “fuel” for smart contracts: an incentive for a miner to execute a smart contract. Quite literally, computation is measured in units called “gas” [18]. The more computationally expensive the transaction, the more Ether it will cost. Forcing individuals to pay gas costs prevents malicious or poorly-designed code from disrupting the network. After all, due to the redundancy of the network, all nodes will need to execute the code of the smart contract.

One of the primary use cases of smart contracts is to create crypto-currencies that run on top of an existing blockchain. These new cryptocurrencies are called tokens, in order to differentiate them from cryptocurrencies with their own blockchains [19]. Not every blockchain-powered application requires a completely new blockchain. Similarly, developing a community of miners to support every blockchain-power application is infeasible. Therefore, Ethereum can be imagined as a blockchain that supports smaller blockchains through the same generalized mining community.

Tokens can be used in a variety of ways. For some applications, a token acts like an in-app currency. A hypothetical application, for example, could take no fees, but require its users to use its token. Therefore, as the application becomes more popular, the demand for the token increases and the token appreciates. The developers own a significant amount of the outstanding tokens, allowing them to sell the tokens at exchanges as profit. Similarly, investors can buy a token to speculate on its future worth. This phenomenon led to the 2017 boom of ICOs, or Initial Coin Offerings, where startups would raise funds by selling off tokens that may have future value [20]. Tokens can also be in more practical ways, such as tracking asset ownership or voting power.

EOSIO

Ethereum is largely considered to be a “second-generation” blockchain because of its introduction of “smart contracts” into the domain. However, there have been a number of newer blockchains who describe themselves as the “third generation” [21]. These emerging blockchains generally aim to iterate on the model provided by Ethereum to create better infrastructures for decentralized applications. Of these, EOSIO has gained the most traction. Launched in mid-2018 by Block.one following a series of Ethereum ICOs that raised over 4 billion dollars [22], EOSIO boasts feeless transactions, supports up to 5000 transactions per second, and consumes significantly less energy than proof-of-work blockchains.

These improvements are largely due to EOSIO’s unique mining protocol. Unlike Ethereum and Bitcoin, where anyone can be a miner, there are only 21 miners at any time in the network. These miners are called “block producers” and are, ideally, geographically distributed.

The mining rewards for each producer depend on their ranking within the top 21, using a consensus algorithm known as proof-of-stake (POS) [23]. Typically, in proof-of-stake consensus algorithms, all coins have already been “minted,” eliminating the mining rewards normally associated with generating a block. In order to support a growing ecosystem, EOS has a flat 5% yearly inflation rate. This inflation rate can be adjusted with proposals that are voted in by the community. Correspondingly, 20% of the newly minted EOS tokens are allocated to the block producers according to their ranking in the top 21. The remaining 80% of these new tokens are reserved for “public” works. To offset this inflationary pressure, Dan Larimer, the CTO of Block.one has suggested EOS can be removed from the market when purchasing resources or domain names on the network.

There are roughly one hundred block producer candidates in the EOSIO ecosystem. Users vote periodically to elect the top 21 producers. Specifically, voting weight decreases every one year, so re-casting votes is necessary to allow for a user to make full use of their voting power. To make this process easier, a user can vote by proxy, or in other words, give their vote to a leader in the community who can appropriately identify worthy block producers [24]. Block producers earn the support of the community by creating tutorials, developing applications, acting as advisories, and producing developer tools.

Furthermore, smart contract execution is not connected to gas. Instead, users “stake” their EOS tokens to the network to purchase CPU, network bandwidth, and RAM. CPU and network bandwidth resources regenerate over the course of three days, while RAM does not [25]. RAM can only be regained by deleting a user’s data in the corresponding smart contract. While RAM and network bandwidth are measured in bytes, CPU resources are measured in microseconds. These resources decrease whenever a user sends funds or interacts with a smart contract.

Altogether, this resource allocation scheme makes it cheaper to maintain and interact with smart contracts on EOSIO compared to smart contract platforms such as Ethereum. EOSIO smart contracts are also written in C and C++, allowing developers to make use of the C++ standard library and Boost. Similarly, rather than create its own bytecode protocol, the EOSIO compiler compiles C++ to WebAssembly [26], an open-sourced bytecode for browsers supported by Apple, Mozilla, Microsoft, and Google [27]. By relying on more established technologies, EOSIO smart contracts are more robust than those supported by Ethereum: they are upgradable [28], they support deferred (asynchronous) transactions [29], and have built-in support for queryable table structures [30].

One of the idiosyncrasies of EOS is its dual key system that involves active and owner keys. An owner key should be kept secret and holds ultimate authority over an account. However, for the vast majority of instances, users make use of their “active” key [28]. The active key is used to sign transactions such as fund transfers and contract actions. A user can also grant smart contracts or other accounts access to use its active key permission. This allows smart contracts to perform actions on behalf of the user. A user can revoke these permissions using their owner key if need be. However, allowing smart contracts to, for example, withdraw a subscription fee automatically from someone’s wallet, mitigates some of the challenges that come with a traditionally “push” based technology. However, it is important to recognize that this two-key system is not intuitive, and many users resort to using the same private key for their active and owner permissions [32]. Even wallets, such as EOSLynx, only generate one private key, setting individuals up for potential theft [33].

Due to EOSIO’s unique mining protocol, many critics have described EOSIO as “not really decentralized” and “not a blockchain” [34]. To an extent, this is true. EOSIO’s more

centralized nature is what allows for its greater scalability. What is more concerning, however, is the apparent collusion between block producers and exchanges [35]. After all, exchanges control the private keys of a number of accounts, allowing them to control the votes of potentially thousands of users (as long as those wallets have positive EOS balances). Similarly, there have been accusations of “vote buying” among Chinese block producers. Block.one recently released a statement that they would use their own EOS to vote against producers guilty of collusion [36].

Existing Decentralized Applications

The decentralized application space is dominated by gambling applications. In part, this is due to users’ desire to evade taxes and circumvent local gambling laws. Regardless, there are a number of decentralized applications worth noting. In this section, we will cover two Ethereum-based applications and one EOSIO application: Civil, Augur, and EOSBet, respectively.

Civil is a marketplace for journalism. Despite a failed ICO in late 2018, Civil presents a solid example of how cryptocurrencies can be used to incentivize community engagement [37]. Civil has two main groups of users: people who simply read journalism and those involved in its production. Civil’s founders split these two groups using a concept they describe as a “Waterline.” Those “below” the Waterline, such as journalists, developers, and community members are the prime users who make use of the CVL token.

There are three primary marketplaces supported by Civil: Newsrooms, Stations, and Fact-checking-as-a-service. Newsrooms are created when CVL token holders pool their funds towards coverage of a specific topic. The token holders can then be involved in the governance of this newsroom by using their voting power to, for example, vote on which journalists will be

involved in covering the topic. Stations are journalist-driven projects catered towards an existing audience — CVL token holders can use their funds to crowdfund the station's operations.

Finally, CVL token holders can stake their tokens to challenge the journalism produced by newsrooms and stations. If the community agrees that the piece(s) of journalism violate the Civil Constitution or present incorrect information (i.e. fake news), then the newsroom or station could face penalties or potential removal from the platform. Meanwhile, the individuals responsible for the challenge are financially rewarded. The Civil platform also involves a number of other initiatives and foundations that are intended to promote free, open journalism [38].

Augur is a prediction market based around the REP token. Users choose an event to bet on, such as the winner of the next presidential election and then proceed to create a market where people can set their bets. Once an event is over, a user can report the outcome of the event by staking their REP tokens. Once a consensus is determined, winnings are accordingly distributed and those who reported the event correctly, or rather, reported with the consensus, receive the REP from reporters who did not report correctly. However, REP is not a token that can be traded on Augur's markets. It can only be earned through participation or by purchasing a set amount on an exchange. After the reporting phase, all fees collected by Augur for a particular market are distributed to the reporters according to the amount they staked when reporting.

Whenever a user creates a new market, they are required to designate a reporter and put up a "no-show" bond. This reporter's initial report, if received within three days, becomes the tentative outcome. If the reporter does not provide an initial report in three days, the report is opened to the market, and the first user to report an outcome receives the original reporter's bond. This first report also becomes the tentative outcome. During the next seven days, any user who holds REP tokens can dispute the outcome. If the appropriate quantity of dispute stake is

raised (which involves crowdfunding), then the disputed outcome is accepted. After a successful dispute, another dispute round is held. To incentivize successful disputes, the bond is designed in such a way as to ensure a 50% ROI for disputing reporters. In the case where a dispute bond amounts to more than 2.5% of all REP, Augur enters a fork state — freezing all other disputes — in order to appropriately handle processing of REP distribution, which at this level involves exterior coordination with exchanges and wallet software developers [16]. An example of the issues that can be caused by Ethereum's lack of scalability.

EOSBet Casino is a decentralized dice-betting game. Although the actual use case seems trivial, the BET token demonstrates how a token can be used as a dividend and an incentive for early adopters to come to the platform. Until all tokens are distributed, users receive 1 BET token for every 20 EOS they wager. Owners of the BET token receive dividends according to their percentage ownership relative to all BET tokens. However, owners of the BET token are not only players but also developers and investors. Their dividends serve as an incentive for them to continue supporting the project.

Chapter 3

Methodology

Because this project intends to provide a real, implemented example of a decentralized application, its methodology is largely inspired by design research, specifically Hevner's three-cycle view of design research [39]. As such, my work can be broken down into three cycles: relevance, design, and rigor. The relevance cycle involves analyzing the current application domain and identifying problems and opportunities within it. The design cycle involves the actual building of "artifacts." The rigor cycle takes the artifacts created during the design cycle and evaluates them according to various methods, theories, experience, and expertise. The next few chapters of this paper correspond to these three cycles.

In the relevance chapter, I discuss the current experience of content creators trying to monetize their work, the common components of a "dapp", and the assumptions and constraints of my project. In the design chapter, I provide an overview of my implemented solution, Submerged. I begin with a higher-level overview of my proposed decentralized model of interaction between content creators and audiences, a breakdown of the corresponding Submerged smart contract, and a high-level mapping of the general application architecture. This last component is especially important because it concerns what aspects of an application should be decentralized and which should not. In the rigor chapter, I evaluate my project according to mathematical methods and theories in user experience.

Hevner's three-cycle view of design research, ultimately, is an iterative process. Thus, this paper concludes with a section regarding future work as well as recommendations for other developers looking to create a decentralized application.

Chapter 4

Submerged – Relevance Cycle

The Problem Domain

Starting in December 2017, YouTube experienced what the community described as the “adpocalypse” [40]. Following pressure from advertisers to prevent brands from being listed as sponsors of, for example, ISIS recruitment videos, YouTube implemented an algorithmic solution that screens videos and removes advertisements from videos that contain “inappropriate” content. In the resulting fallout, videos that simply featured controversial topics or vulgar language were flagged and demonetized, even in informational contexts. Philip DeFranco, for example, a prominent YouTuber who runs daily news digests, claimed to have experienced a drop of 80% in ad revenue [41].

YouTube does support an appeals process that allows creators to challenge demonetization on a case-by-case basis. However, considering that videos garner most of their traffic within the first 3 days of going live, the window to generate the most ad revenue has already passed by the time the appeals process concludes. Therefore, many creators see ad revenue as an even more unreliable and unpredictable source than it once was, and have turned to other solutions to monetize their work. Top Youtubers can rely on merchandise sales; most rely on some sort of crowdfunding.

The most popular third-party solution is a service called Patreon, which allows audiences to pay a monthly subscription to a content creator (YouTuber, podcaster, blogger, artist).

However, this subscription, in reality, is more of a donation. It is not a payment for a service, meaning that a creator is not obligated to actually create content. Most Patreon channels offer various tiers of “subscription.” For example, three dollars a month for six months may earn the subscriber a unique sticker that is mailed to their house. Six dollars a month for six months may earn them a signed poster. In most cases, the benefits of subscribing are mainly these kinds of “soft” benefits. Creators can also charge a fee per unit of content produced (e.g. per video), but in turn, monthly payments can vary wildly depending on the creator’s production schedule [42]. On average, Patreon’s fee ranges between 10 and 15 percent of a creator’s earnings [43].

To compete with Patreon and Twitch, YouTube launched its own “Channel Membership” program that allows audiences to become “members” of a creator’s channel for \$5 dollars a month. In turn, subscribers can earn badges as well as other goods from the creator. However, this feature is restricted to creators with more than 30,000 subscribers, is set at a fixed price of 5 dollars, and entitles YouTubers to only 70% of the funds they raise [44].

This project is largely inspired by the services offered by Patreon but aims to improve on the existing model by making use of blockchain technology to reduce transaction fees, formalize the relationships between content creator and subscriber, and create a system of financial incentives that promote engagement on the platform.

Technological Constraints/Dependencies

Most user-facing decentralized applications require users to sign transactions using their private keys. There are many specialized wallets that offer users this functionality. This project supports ScatterJS, a wallet that supports Ethereum, EOS, and TRON smart contract interactions.

Simultaneously, the wallet features a built in-exchange that allows users to exchange tokens on the same blockchain without paying a fee. Inter-blockchain currency exchanges, such as BTC to EOS, cost .5% of the total transaction. ScatterJS also supports the creation of identities that can serve as OAuth and also provides both Desktop and Mobile support. For these reasons, ScatterJS is by far the most popular wallet for EOS applications and is therefore supported by this MVP.

Storing all relevant application data on the blockchain is not feasible, nor a smart design decision. This application, by its inherent nature, deals with a significant amount of video data, and the decentralization of content-hosting is not one of the goals. Therefore, the application depends on existing content-hosting services such as YouTube and Vimeo. Thankfully, YouTube allows creators to restrict the domains where their videos can be embedded. Therefore, in our use case, where creators may desire to create a video exclusive to Submerged, they can customize the video's privacy settings to achieve the desired effect.

Chapter 5

Submerged – Design Cycle

Choosing a Blockchain

The two most dominant infrastructures for smart contracts are Ethereum and EOSIO. This project was implemented using EOSIO for a variety of reasons, but most of all, cost. Ethereum requires a user to pay gas costs for every transaction (roughly 25 cents), while EOS does not. Considering that one of the main goals of this project is to increase the amount of money creators can take home from crowdfunding, this difference alone makes EOS more attractive. Similarly, if one examines the amount of dapp activity across blockchains, EOS dapps outperform every other protocol, with more users and higher engagement. It seems natural to assume that this in large part due to users not needing to pay a gas fee for every transaction [45]. Simultaneously, writing smart contracts for EOSIO is a more manageable experience compared to Ethereum. A developer can make use of virtually any existing C or C++ library. Contracts, too, are not completely “immutable” and can be upgraded. Ethereum contracts, on the other hand, are limited to the capabilities of Solidity, which is still a young and unrefined programming language.

A New, Formalized Relationship Between Creators and Audiences

Table 1 Submerged Terminology

Term	Definition
Channel	A creator’s “page” on Submerged. This is where creators deliver their content and can make update posts.
Project	A single piece of content produced by a creator (a podcast, a video, a song, and so forth).
Submerged Foundation	An elected body composed of 5 representatives from the community. 1 seat is reserved, at least initially, for the developers.

On the Submerged platform, a creator’s channel cycles between four stages — *declaration, fulfillment, reporting, and settlement* — every 30 days. At the beginning of the cycle, in the declaration stage, creators declare the quantity, type, length, and deadline of the monetizable projects they will produce in the coming 30 days. For example, a creator may declare she will create two 15-minute episodes of her podcast due the 10th and 18th of the coming month, respectively. Unless specified by the creator, this declaration will recur at the beginning of the next cycle. Declarations must occur before the beginning of the billing period.

The declaration stage is first and foremost, a promise to a creator’s community and an obligation to fulfill. It specifies exactly what a subscriber can expect to receive in turn for a subscription. Although it may seem strict to force creators to adhere to a schedule, other platforms, such as YouTube, already have built-in biases in the recommendation algorithm for creators who upload multiple times a week. Similarly, other subscription-based services, such as newspapers, magazines, and even Netflix, promise new content with some regularity. Therefore,

it is not unreasonable to expect regular content in exchange for regular payment.

After a creator has declared their projects for the upcoming cycle, they must fulfill, or rather deliver, the content before the deadline they specified. If the creator does not deliver the specified content on time, then the project is automatically forfeited for that billing cycle. However, creators can petition their audience for a time extension three days prior to the due date.

If the content is delivered on time, then reporting opens for a set period of time on that specific project. If a project is delivered more than three days before the deadline, then reporting lasts until the day of the deadline. Otherwise, reporting lasts for three days. Audience members vote on whether this piece of content satisfies the declaration provided by the creator. If more than 25% of all subscribers are dissatisfied by this content, meaning they do not believe it satisfies the requirements set by the creator, then the project is rejected. Any campaign, such as time extensions, operates on the same 25% principle.

This benchmark was adapted from the principle of a net-promoter score (NPS). NPS is a measure used to determine how likely a person is to recommend a service to a friend or colleague, but more generally, provides a solid indicator of a customer's satisfaction with a product [46]. An NPS of 0 represents a neutral disposition, where an NPS above 50 is considered above average. Submerged uses this measure rather than a simple majority because it assumes that a person who does not report is satisfied. At the end of the reporting phase, a project is either accepted or rejected by the community.

A creator can appeal any project that is rejected by the community to the Submerged Foundation. The Submerged foundation can then reverse the decision of the community if deemed appropriate. Simultaneously, any subscriber can launch an appeal against a creator by

staking their tokens and opening an appeal to the Foundation.

After the last project of the billing cycle is fulfilled or rejected, a Channel enters the settlement phase. During the settlement phase, a percentage is calculated based on the number of fulfilled projects over the total declared projects for the billing cycle. Then, a creator receives the corresponding percentage of the total amount subscribed to them during that billing cycle. In other words, if a creator has a 100% fulfillment rate for a given cycle, they receive 100% of the funds subscribed to them. Otherwise, subscribers are credited the difference. This process is represented visually in Figure 3 below.

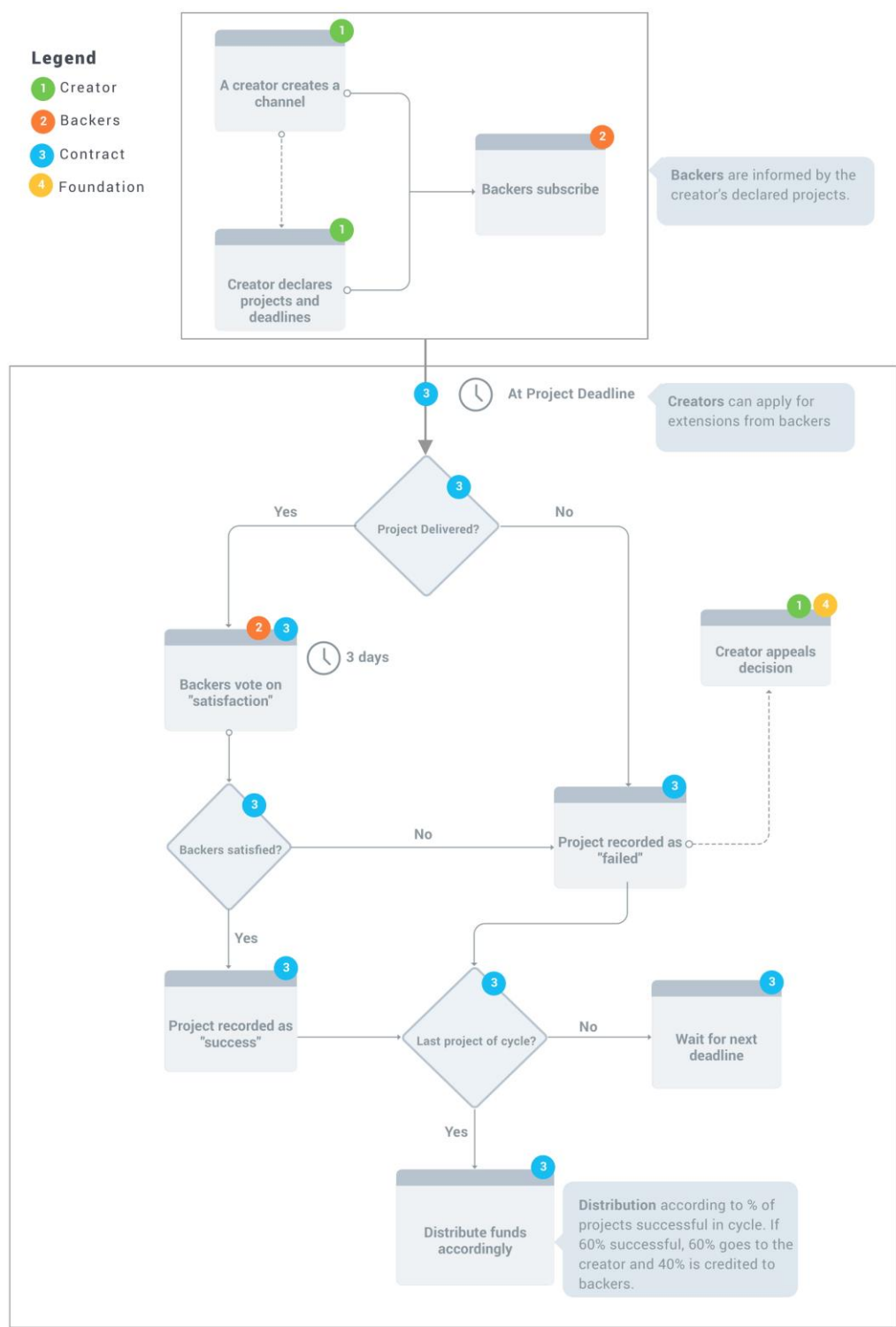


Figure 3 Diagram of Submerged Model

The SUBM Token

All subscriptions are currently fulfilled with EOS. In the future, this may be replaced by a third-party stable coin. The proposed Submerged model does, however, support a token (SUBM). The SUBM token is intended to incentivize engagement on the platform and attract early adopters. Every quarter, the fees collected by Submerged (in EOS) are distributed as dividends to token holders. The following token allocations are only tentative. To determine the proper allocation would require an economic model based on the success of a token sale and knowledge of the size of the initial community.

Tokens can be purchased or earned through activity on the platform. For example, correctly reporting satisfaction earns the user 1 SUBM token. For every fulfilled project, a creator receives 5 SUBM tokens per subscriber. If a creator is on a “streak,” or has fulfilled every project for at least three months, they earn 7 SUBM tokens per subscriber. SUBM tokens are also used to compensate members of the Foundation. Making an appeal to the foundation requires 1000 SUBM. Requiring users to stake their tokens to appeal prevents an individual from spamming the foundation and limits appeals to those with a proven dedication to the community.

The Submerged Foundation

The Submerged Foundation is composed of five representatives elected by the community. Running for a seat requires candidates to stake SUBM tokens, proving their contributions, either financial and/or participatory. The winners of the election receive the staked tokens of those who were not elected. This elected body votes on appeals brought forward by both creators and audiences alike. The staked tokens of each appeal are evenly distributed across

all members of the Foundation after a decision is made. If a member is inactive, meaning that they have not voted for more than 10% of all appeals in a quarter, their seat will be put up for election. The motivation for pursuing a governance model based on elected representatives, rather than a completely democratic voting process, is twofold. First, audiences may disapprove of a controversial video. It only takes an impassioned minority to qualify a video as “unsatisfactory” (25%). After all, it is possible that those alienated by a creator’s content would be more likely to share their disapproval than those of more neutral disposition. In such cases, the community’s “rejection” is not due to a failure to deliver the content on time, or fulfill a promise on content length. Therefore, rather than democratizing the solution, which would draw attention from those who already expressed their dissatisfaction, representatives may be able to provide a more objective decision.

Simultaneously, audiences may disclose content creator violations. For example, a user may report a creator who uploads copyrighted content. In such instances, audience approval may be very high, but the monetization of proprietary content legally threatens the existence of Submerged as a whole. Thus, audiences, or concerned individuals, can bring forth appeals. If an appeal against a content creator is successfully voted through, then the winning party is rewarded with SUBM tokens from the Submerged developers. Content creators who appeal against their audiences receive the funds put forth by their audiences for the initial subscription.

The Submerged Smart Contract

Dan Larimer, the CTO of Block.one, advises developers to create one contract rather than separate concerns across multiple smart contracts [47]. One of the chief reasons for doing so is to

avoid network costs and make it easier to manage RAM and CPU costs. Following his guidance, the Submerged blockchain back-end is one monolith contract. It supports creating channels, declaring projects, audience reporting, fund distribution, timing, and applying for extensions.

The two features that are not included in this implementation are the SUBM token and the appeals process/Foundation. All tokens on EOSIO are handled by the eosio.token contract, a contract natively supported on EOSIO. Similarly, there are existing tools, such as EOSDrops, that handle the process of “airdropping” tokens to specific users. Furthermore, since this distribution would only take place until all tokens are distributed, this feature is not a permanent aspect of the contract. The appeals process is not included in this implementation because it requires a significant amount of input from content creators. Civil, for example, bases its own “constitution” on conversations with free press activist organizations and institutions such as NPR. Furthermore, because the Foundation is inherently rooted in user deliberations, it would be difficult to evaluate the efficacy of such a form of governance given the scope of this project. Similarly, it is assumed that the appeals process would only need to be considered for a small proportion of all projects declared on the platform.

In terms of deployment, only one instance of the Submerged contract is uploaded to the network. All Submerged users make use of the same contract. This is made possible through the use of multi-index tables that are supported natively on EOSIO. Tables are maintained as long as a user pays for the RAM they require. On EOSIO, tables can be additionally scoped to help prevent concurrency issues and separate records. Put simply, a table may relate to a specific content creator or the whole application.

Table 2 Multi-Index Tables in the Submerged Contract

Table	Scope	Purpose/Attributes
Channels	contract	The “channels” table holds general information regarding each channel, such as price per billing cycle, the number of declared projects, and the number of fulfilled projects.
Channel Subscriptions	creator	Each “channel subscriptions” table is tied to a specific channel. It lists all the subscribers for a channel and whether or not they have paid their subscription. This table is especially important when users opt-out of sharing permissions to auto-recur their subscription.
Projects	creator	The “projects” table holds information related to each project, such as its current status (failed, payment pending, complete, etc), and its promised parameters.
Polls	creator	The “polls” table carries information related to any vote, be it a creator asking for an extension or audience members validating content.
Users	contract	The “users” table contains entries regarding a user’s subscriptions and their settings, such as whether auto-recur is allowed.

There are three components of the Submerged contract that are uploaded to the network: the WASM (C++ compiled to WebAssembly) file, the ABI (Application Binary Interface) file, and Ricardian contract. The ABI file is responsible for connecting the binary code (of the WASM file) to specific human-readable commands. These commands are called through an RPC (remote procedure call) API or command line. The Ricardian contract is a form of documentation, technically required by Block.one (but not enforced) for any smart contract uploaded to the mainnet, that informs a user of the purpose, function, and consequences of interacting with a contract . For example, in our case, if a user wanted to create a channel using the Submerged contract, they could look up the Ricardian clause for the “create channel” command.

The EOSIO C++ compiler takes one .cpp file as input and compiles it into a smart

contract. Therefore, writing the smart contract to support all the features listed above required unconventional C++ design choices. First of all, most of the contract's logic was placed in header files, using classes that were defined, rather than just declared. This not only made development more manageable, but ensured that all the logic and data models could still be aggregated into one cpp file and, consequently, one smart contract. Therefore, the actual cpp file can be imagined as a façade for an underlying monolith system. The pseudo-code below outlines how the cpp file takes shape:

```
#include EOSIO library
#include common structs, multi index tables, and controllers
#include header file for Submerged contract

// override Submerged header file, use ACTION annotation
ACTION submerged::openchannel( ...parameters... ) {
    the_channel_controller.open_channel(...parameters...)
}

ACTION submerged::fulfillproject( ...parameters... ) {
    the_project_controller.fulfill_project(...parameters...)
}

/* repeat the pattern above for every action */

/*
EOSIO custom ABI dispatcher (to listen for transactions from eosio.token)
*/
```

Figure 4 Contract Pseudocode

Unlike Ethereum, where Ether is attached with every transaction (to support gas), all

token transactions occur in the jurisdiction of the eosio.token contract. To connect with the actions (sending, receiving tokens) of the eosio.token contract, the Submerged smart contract includes a custom ABI dispatcher.

The ABI dispatcher connects action names to their respective methods in the Submerged smart contract. However, we can also add custom logic to allow for the smart contract to listen to events in the eosio.token contract. Therefore, the Submerged contract has its own “transfer” action that reacts any time the Submerged contract is the receiver or sender of funds in the eosio.token contract. The transfer action of the smart contract then intercepts the parameters of a successful transfer, and, by parsing its memo, can appropriately allocate funds. Therefore, the actual transfer of funds between subscribers and channels, and any subsequent withdrawals, all make use of the eosio.token contract already implemented on the network.

Blockchain technologies, by their inherent design, are push based. Every transaction must be authorized by a user and fired by a node. EOSIO works around the limitations of a being a push-based technology by supporting deferred actions and a two-key system. Deferred actions, as the name suggests, are transactions that fire after a certain period of time. This time delay can be set by a user when they push a transaction, or as in this case, by the contract itself. Every deferred transaction must fire within 45 days. The Submerged contract makes generous use of the deferred action capability to set deadlines for its various actions. This way, the smart contract does not depend on a server to push actions at various time intervals, allowing it to be self-contained and more decentralized. However, deferred actions are not guaranteed to fire. If a deferred action is set for 3 hours into the future and the contract (or the CPU/RAM payer) does not have sufficient resources, then the action cannot be called. As recommended, the Submerged contract supports certain actions to be called manually. However, when deploying, it would be of

utmost importance for the developers to ensure that enough EOS is staked to cover EOS resource costs.

Table 3 Smart Contract Actions

Action Name	Called By	Description
version	any user	Prints the version number of the contract to the console
open	creator	Allows a user to open a channel and begin receiving subscriptions
transfer	subscriber	Whenever a subscriber transfers money to the Submerged contract, the contract records the transaction and parses the memo to allocate the funds to the creator (can also be used to credit an account)
recur	contract	The contract calls this method every billing cycle to allow for a “real” subscription to occur
initproject	creator	Called by creators to “declare” a project and assign a due date
fulfill	creator	Called by creators when they have “fulfilled” their project
fail	contract	Called by the contract if a project is not delivered before the deadline
closevoting	contract	Called by the contract to close the vote on an extension or satisfaction measure
applyforext	creator	Called by the creator to change a deadline of a project
vote	subscriber	Pushed by subscribers to voice their opinion on an extension or satisfaction measure
paychannel	contract	At the end of the billing cycle, the contract calls this method to credit the channel
creditsubs	contract	At the end of the billing cycle, the contract calls this method to credit subscribers if projects requirements have not been satisfied
unsub	subscriber	Subscribers call this method to unsubscribe from a channel
withdraw	any user	Allows a user to withdraw the funds that are credited to their account and transfer them to their wallet

After pushing their initial subscriptions, users can give the Submerged contract permission to recur the payment on a 30-day basis. This functionality is achieved by the user

giving permission to the smart contract to use the user's active permission. This allows the smart contract to withdraw the appropriate amount of EOS tokens from the user's wallet every 30 days: like a real subscription. Otherwise, users need to push the transaction themselves every 30 days. If the user is ever concerned about these permissions, they can revoke access using their owner key.

Who Pays? Users or the Developers?

One of the benefits of decentralization is moving costs away from the smart contract to the users. For example, a user could pay for their own RAM fees. In our case, it would be cheaper for the developers to require a content creator to pay for the cost of maintaining their project declarations and the subsequent reporting phase. Although this functionality can be beneficial for applications such as a games, where a user pays for the storage of their inventory items on the blockchain, it proves to be difficult to handle in our situation where multiple parties rely on the same data tables. Due to the way multi-index table permissions are structured and due to the reliance on deferred actions, delegating costs to users is not ideal. If the content creator does not have enough RAM resources, then users' subscriptions may not go through. This not only disrupts the whole ecosystem but also demands that content creators worry about maintaining proper resource allocations.

By default, on the EOSIO network, all actions called by a user are paid for using the user's CPU and bandwidth resources. Only RAM costs can be delegated. However, all subsequent actions called by the contract are paid for by the contract. In the case of the Submerged contract, nearly all of the most computationally intensive actions are called by the

contract, thus decreasing the number of the user's resources consumed. Regardless, building on EOSIO requires users to have some EOS staked the network. Thankfully, the RAM costs are completely covered by the contract, meaning that the only user resources consumed are re-generable. The details regarding these upfront costs are taken into account during the rigor cycle.

The Submerged Application

Submerged's overall architecture can be broken down into 4 layers: front-end, server, database, and blockchain. The frontend-layer is composed of a single-page application built with React/Redux that interacts with the ScatterJS API, EOSJS API, and the server layer. The server is built using Nest.js, a Node.js framework built on Express that supports static typing through TypeScript. The database, which connects to Google Firebase through the server, acts as a NoSQL database for data not related to blockchain transactions. Finally, the blockchain layer is composed of the Submerged contract components listed above. It manages all logic that decentralizes the financial aspects of Submerged. The sequence diagram below (figure 5) describes how these four layers (and ScatterJS) interact when a user subscribes to a channel for the first time.

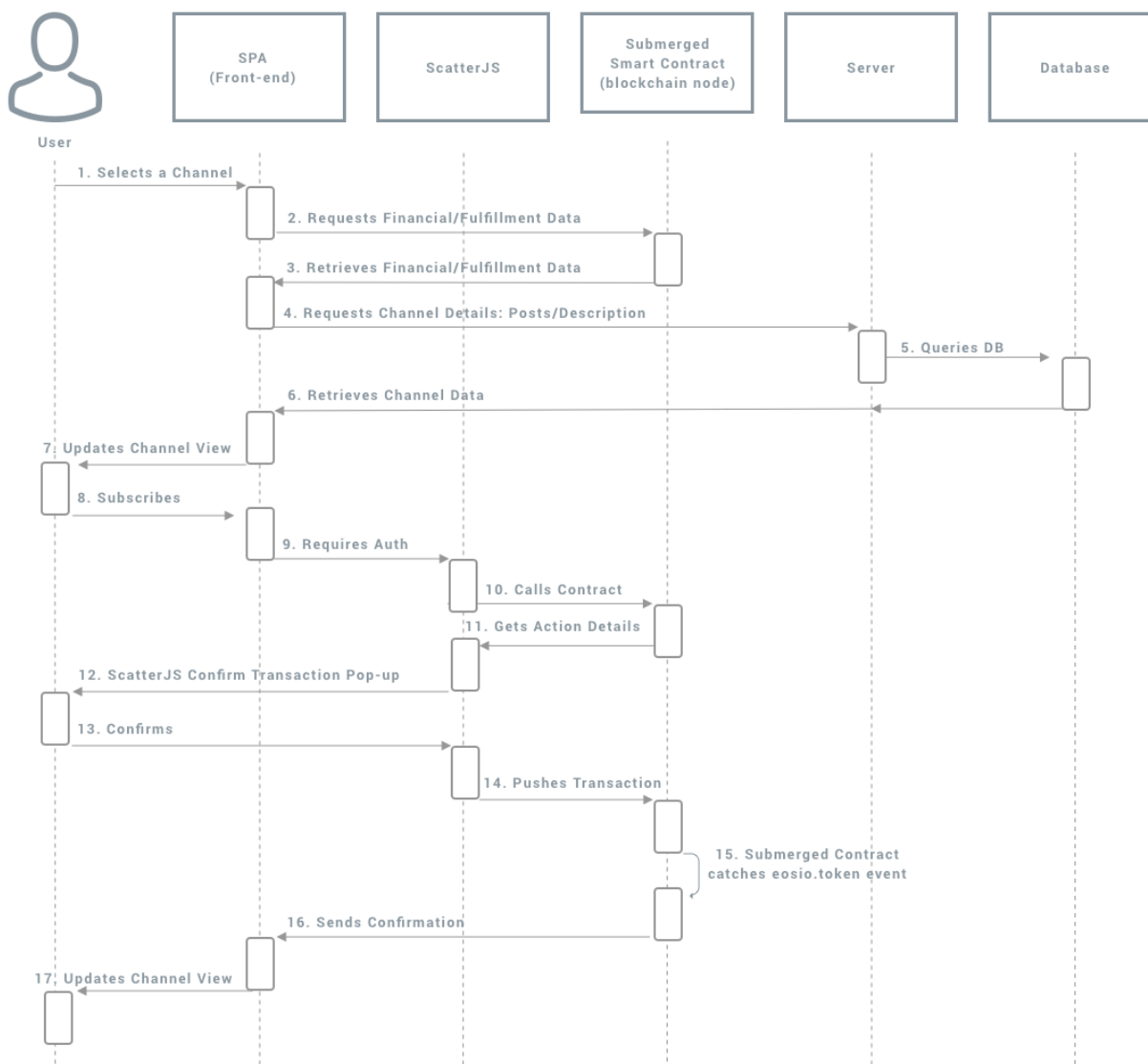


Figure 5 Submerged UML Sequence Diagram

The most notable design choice, architecturally speaking, is the separation of financial and social data. The Submerged smart contract only stores data relevant to financial transactions, fund distributions, and governance. All other forms of data, such as user profiles, posts, and icons, are stored using “traditional” databases. After all, RAM is a finite resource on the EOS

network. For the sake of cost savings for creators and audiences alike, it is best to minimize the data stored on the chain to the bare minimum necessary.

The Submerged Application Walkthrough

The purpose of this section is to include screenshots from the implemented application to illustrate how it works, and how the front-end serves as an abstraction of the smart contract.

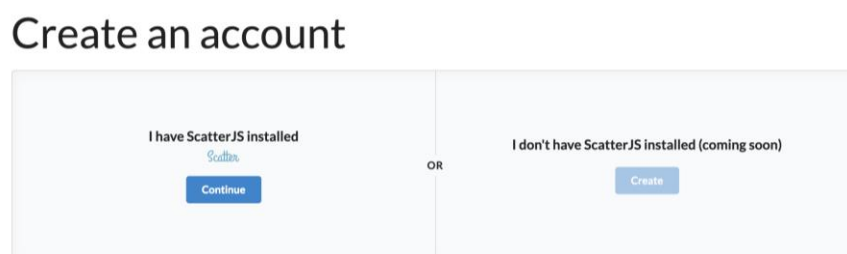


Figure 6 Creating a Submerged Account

Before a user can interact with Submerged, they need to create an account with their Scatter wallet. Scatter allows users to create identities populated with the user's demographic information, and allows applications to make use of this information as a kind of OAuth. At the current moment, Submerged requires users to use a Scatter a wallet, and thus requires users to sign-up with Scatter (Figure 6). Figures 7 and 8 depict how this sign-up takes place within the Scatter app.

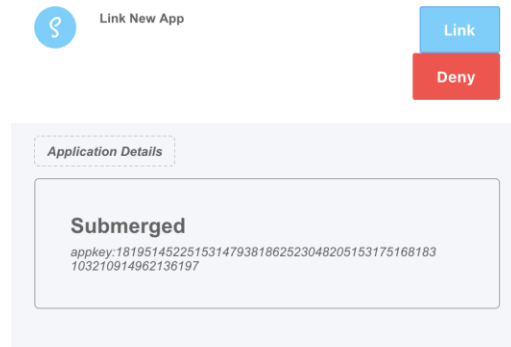


Figure 7 Linking Scatter to Submerged

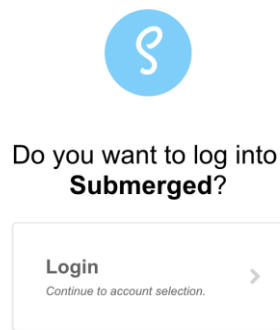


Figure 8 Login into Submerged

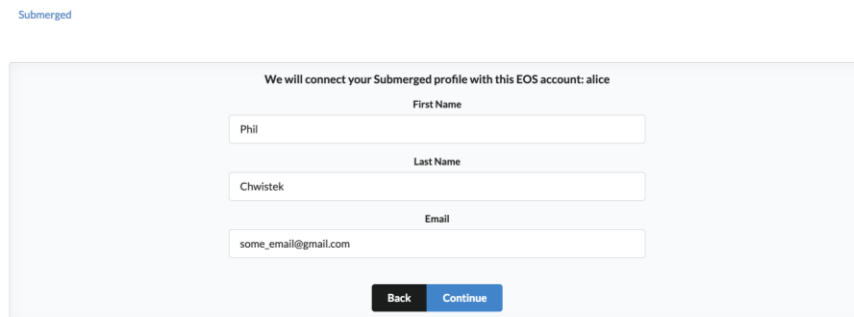


Figure 9 Scatter Identity

Figure 9 demonstrates how Submerged pulls information from the user’s Scatter identity, and also displays which EOS account the user’s actions will be associated with.

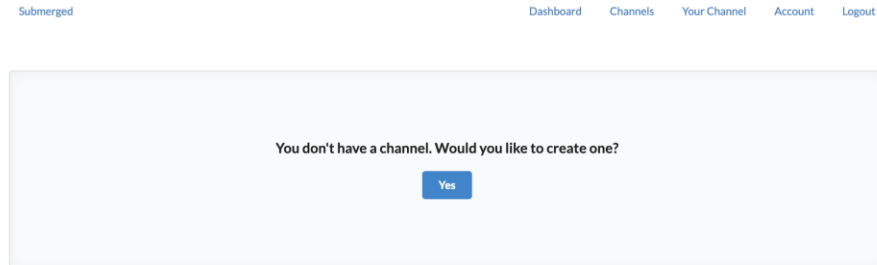


Figure 10 Creating a Channel

Submerged users are not required to create a channel, but to be able to receive funds, users must create a channel that contains information that is split across the blockchain and traditional database layers.

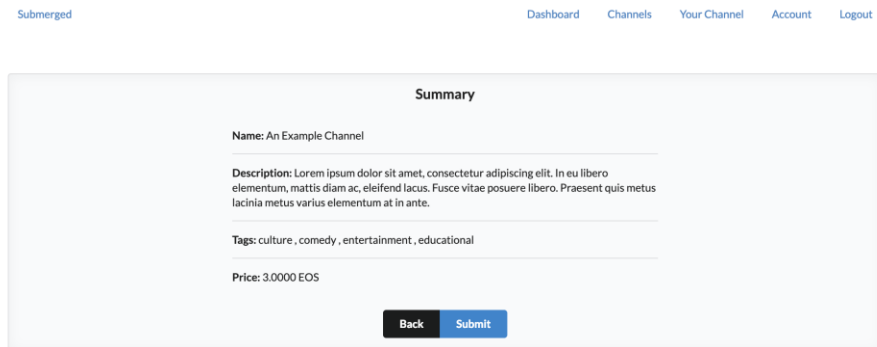


Figure 11 Channel Summary

Figure 11 demonstrates the summary page after the user fills out the form for creating a channel. The tags and channel name fields help other users search for a specific channel. The price field is related to how much a content creator will charge per billing cycle.

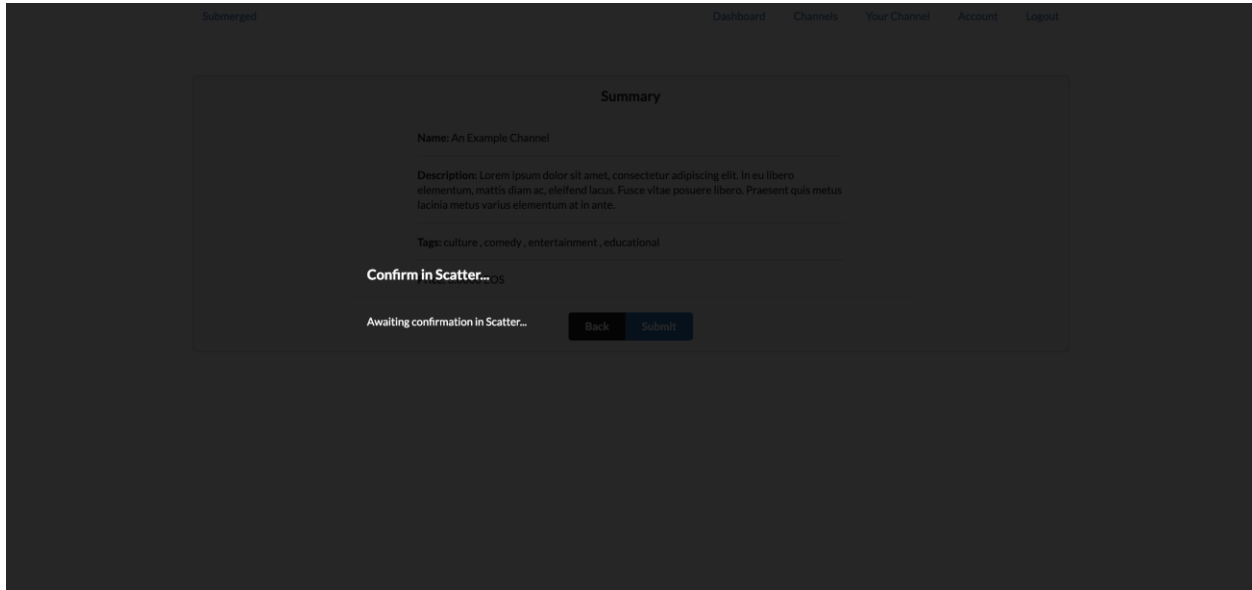


Figure 12 Scatter Modal

Figure 12 demonstrates the in-app modal that informs the user to change windows to the Scatter wallet. Typically, Scatter window modals are “brought up to front,” but this in-app modal ensures that the user is aware of the needed Scatter signature.

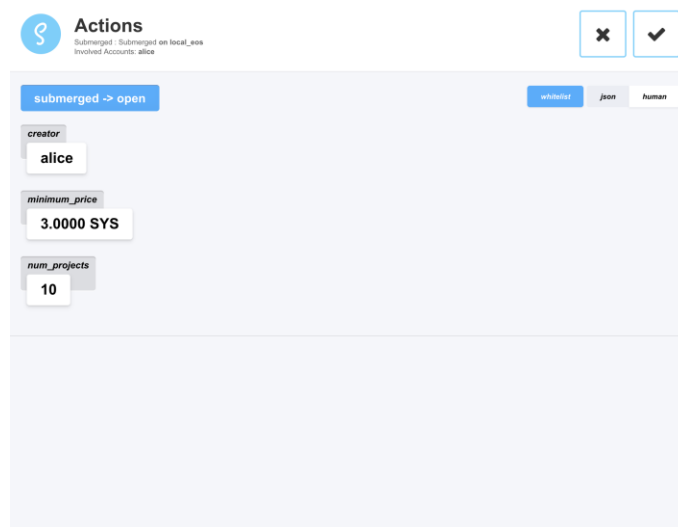


Figure 13 Scatter Create Channel Confirmation

Figure 13 displays the Scatter modal that informs the user of the transaction they are about to sign. All the necessary parameters are grabbed from the Submerged application and are displayed for the user. In our case, this example is creating a channel under the “alice” account, where the maximum number of projects in a cycle is 10 and the subscription is priced at 3 EOS. The user proceeds by clicking the check button in the upper right-hand corner.

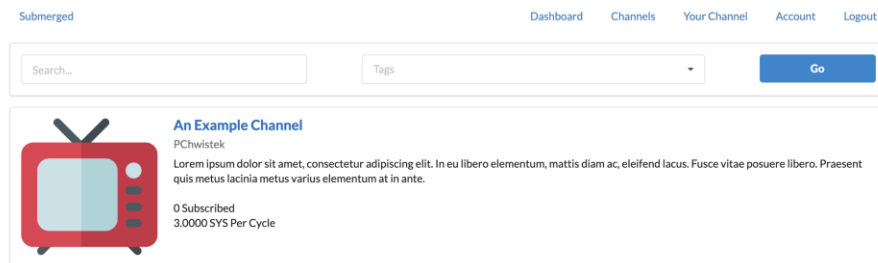


Figure 14 Channel listed in Channels List

Once accepted by the smart contract, a channel is now officially opened on Submerged. A user can find channels to subscribe to under the “channels” tab.

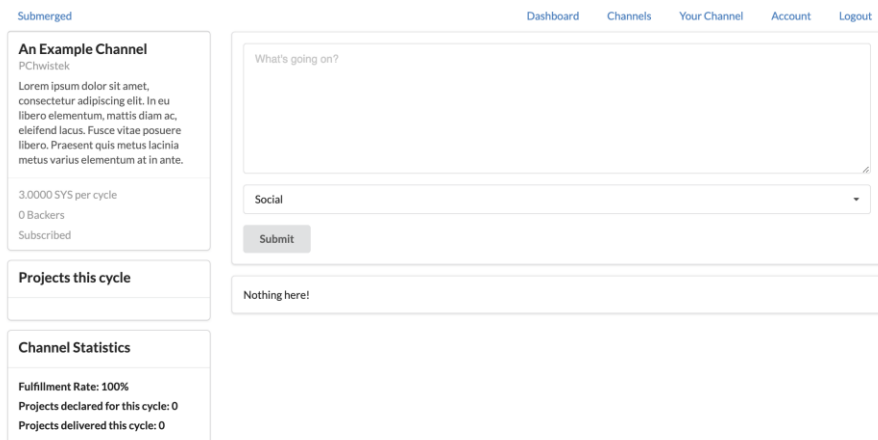
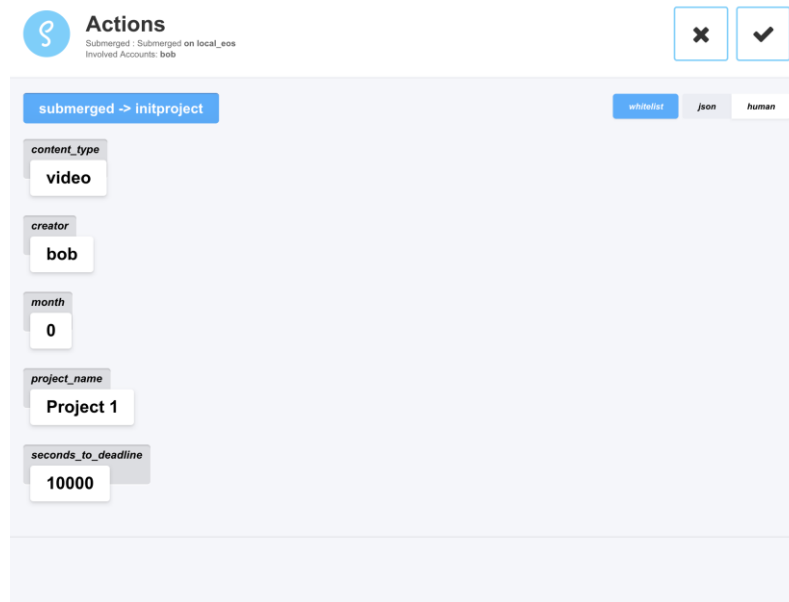


Figure 15 A User's Channel View

When a user views their own channel, they see a form where there they can post relevant information to their audiences. There are four types of post: declaration, delivery, social, and extension. Social posts are regular posts with no underlying connection to the blockchain. They can be imagined, conceptually, as the same thing as Facebook posts. Declaration, delivery, and extension posts all relate to a creator's projects.



The screenshot shows a web interface for declaring a project. At the top left, there is a logo with a stylized 'S' and the text 'Actions'. Below the logo, it says 'Submerged: Submerged on local_eos' and 'Involved Accounts: bob'. To the right of the logo are two buttons: one with an 'x' and one with a checkmark. Below this is a form titled 'submerged -> initproject'. The form has three tabs: 'whitelist', 'json', and 'human'. The 'whitelist' tab is selected. The form contains several input fields: 'content_type' with the value 'video', 'creator' with the value 'bob', 'month' with the value '0', 'project_name' with the value 'Project 1', and 'seconds_to_deadline' with the value '10000'.

Figure 16 Declaring a Project

Once a user fills out the declaration post, they must sign the transaction once again using Scatter.

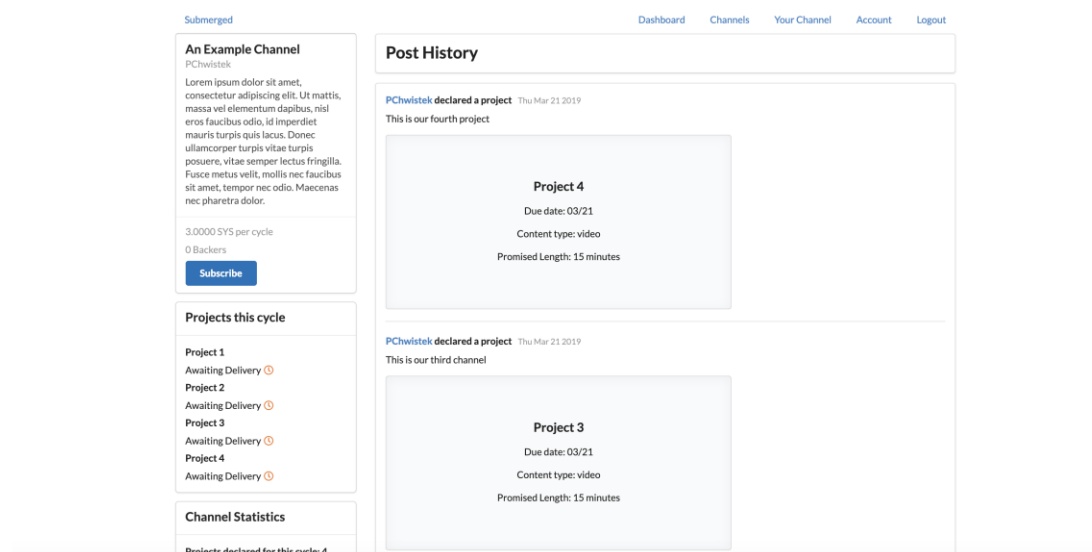


Figure 17 Feed Populated with Declared Projects

Figure 17 depicts a Channel's page from the perspective of a potential subscriber. Here, there are 4 project declarations. In the bottom left-hand corner is a summary of the projects that a content creator has declared and their current status.

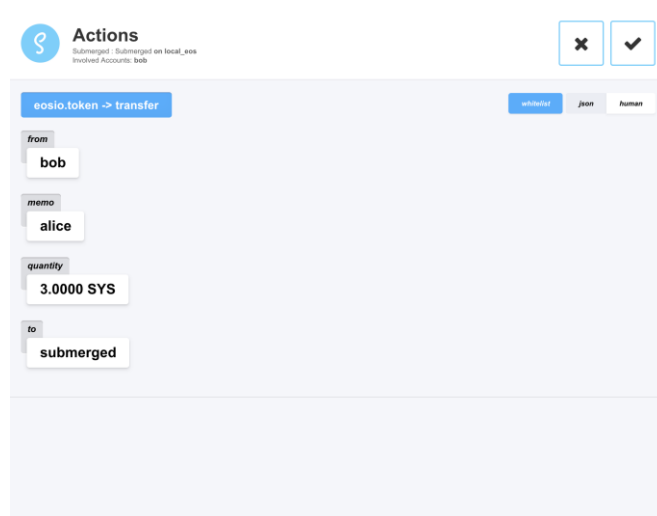


Figure 18 Scatter Subscription Confirmation

When a user subscribes, they send their funds using the existing eosio.token transfer contract. The memo contains the account name to which the channel belongs.

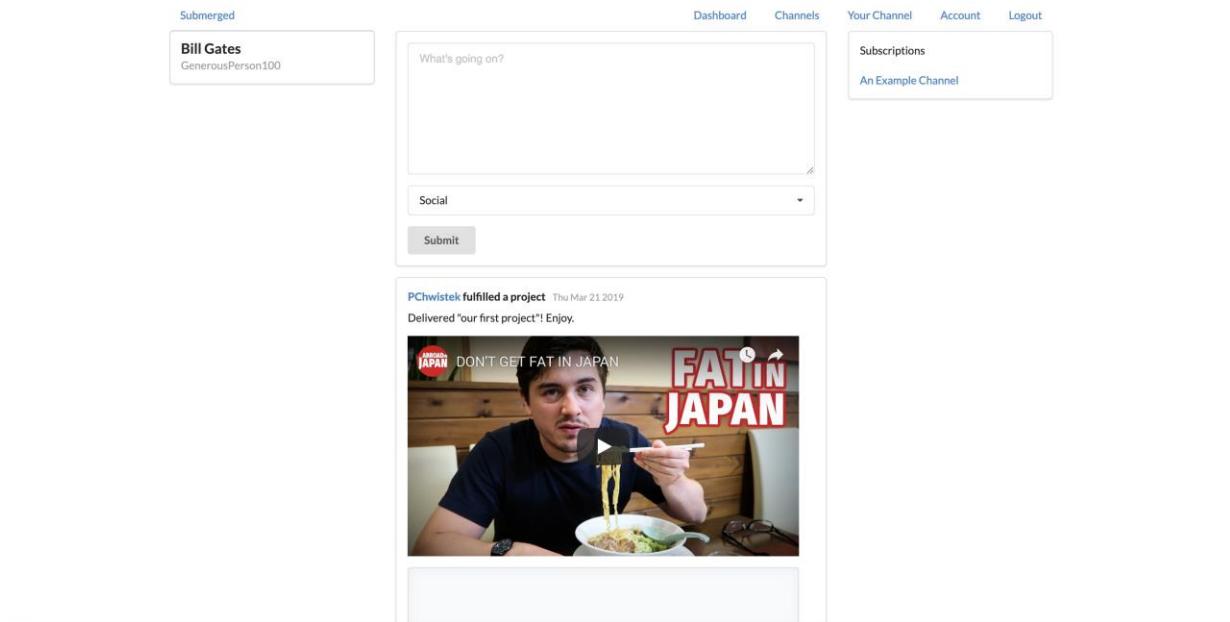


Figure 19 Dashboard View Post Subscription

Figure 19 depicts the dashboard of a subscriber. The right-hand side contains a list of their current subscriptions, and their social feed consists of posts made by the content creators they are subscribed to. The form limits all their posts to “social” posts as they do not have a channel.

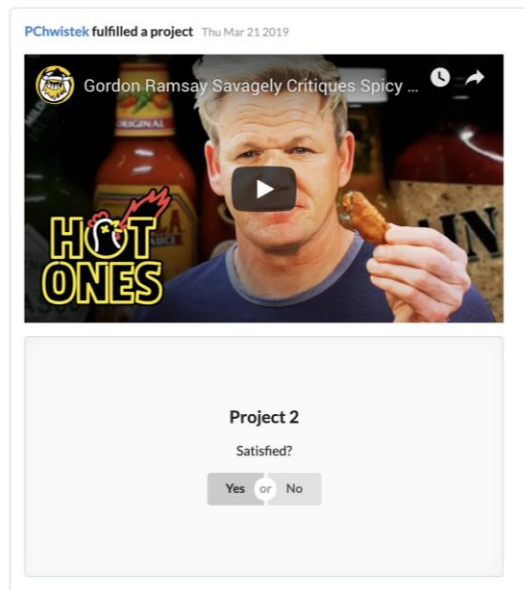


Figure 20 Feed Event For Delivered Project

The screenshot above shows the prompt that appears beneath delivered projects that are in “payment pending” status. It is here that a user votes on whether or not the content satisfied the original promise made by the content creator. Figure 21 is the corresponding Scatter confirmation.

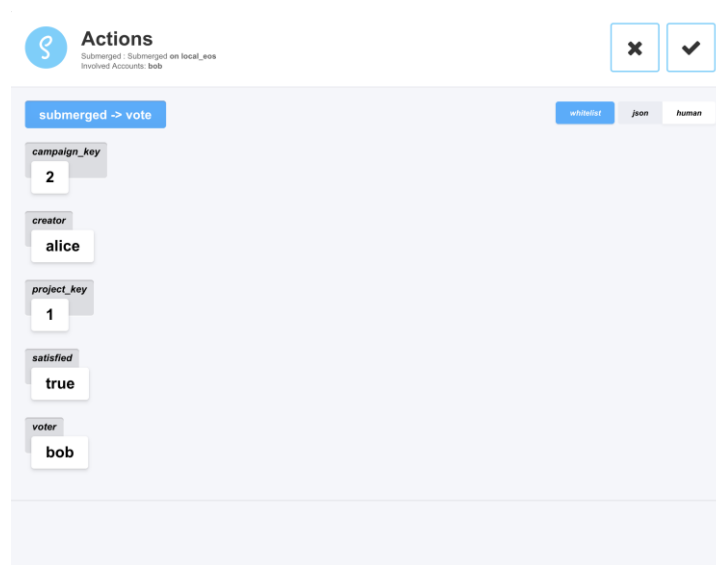


Figure 21 Scatter Vote Confirmation

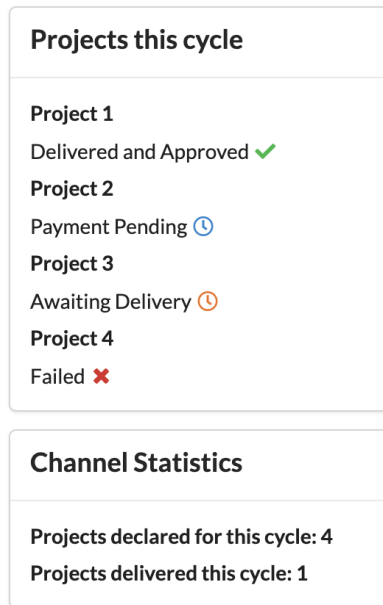


Figure 22 Channel Summary

Figure 22 depicts a channel summary with four projects with four different statuses. A project with the “payment pending” status is a project that has been delivered, but is currently taking votes from the community to assess its validity. A project “awaiting delivery” has been declared but not delivered. A “failed” project is a project that was not delivered on time or was rejected by the community. A project that has been delivered and accepted by the community is marked as “delivered and approved” and will be counted for the creator at the end of the billing cycle. Figures 23 and 24 show the modal detail view that is accessed when a user clicks one of the projects in the summary or one of the posts in the feed. Doing so allows a user to check on a project regardless of whether they’ve clicked on the declaration or delivery of a project.

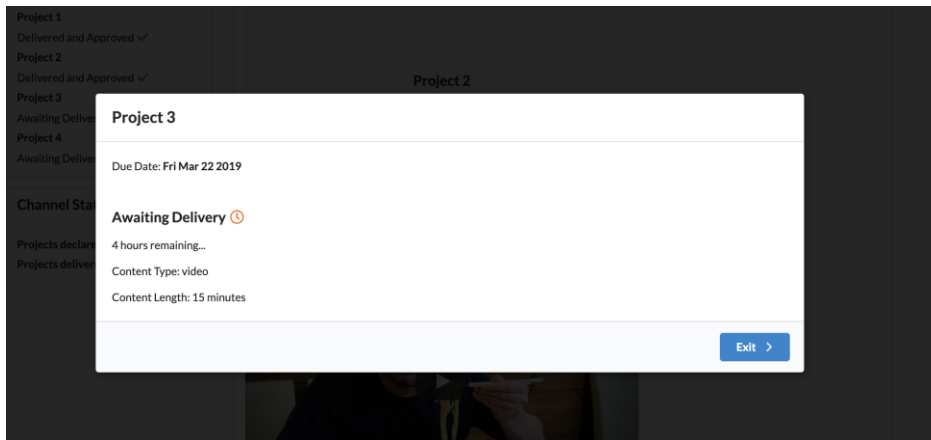


Figure 23 Project Detail Modal with Incomplete Project

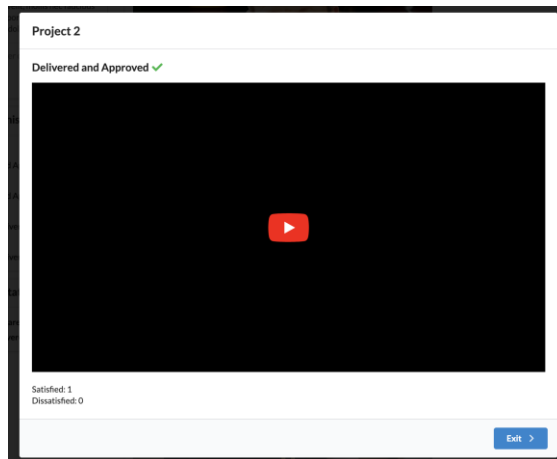


Figure 24 Project Detail Model with Delivered Project

Chapter 6

Submerged – Rigor Cycle

Methodology for Estimating Contract Costs

The data listed below comes from the logs produced by a local instance of the EOS blockchain on a 2018 MacBook Pro. The RAM and bandwidth costs are identical to those that would occur on a live network. The CPU costs, however, which are measured in microseconds, are likely to be overestimated. EOSCharge, a service provided by EOS New York, shows that the eosio.token transfer, on average, takes 1104 μ s, while the eosio.token transfer on the local instance takes between 1300 μ s and 2000 μ s [48]. This difference can be attributed to the cloud infrastructure provided by block producers, which almost certainly offers more computational power than a single MacBook. Each measure of an action's length, in microseconds listed below, was calculated by taking the average of 10 trials per action. The size of structs, measured in bytes, was provided by the sizeof method from the stdio.h library. The current EOS resource prices were provided by the EOS Resource Planner, another service provided by EOS New York. The following evaluations assume that 1 EOS is equal to the price of \$3.69.

Estimating User Costs

Unlike most other blockchains, creating an account on the EOSIO network is not free. The recommended resource allocations for a basic account are 4 KiB of RAM (RAM costs vary

based on market factors), .15 EOS dedicated to CPU, and .05 EOS dedicated towards network bandwidth. Furthermore, an account must be created by an already existing account. Various block producers offer open-source account creation services for a small fee. The most popular account creation service, offered by EOSVibes, charges .7 EOS (roughly \$2.60) for account creation with the above resource allocations. To determine a conservative estimate of the upfront costs a user would need to incur to create an account with enough resources to use Submerged, we can rely on the following narrative:

Alice creates an EOS account using EOSVibes, opens a Submerged channel, declares 4 projects, fulfills 4 projects, subscribes to 5 creators, and reports on 5 projects delivered by the creators she is subscribed to, and adds credit to her account.

The above narrative assumes that these actions are all completed within a 24-hour time frame and that we want to avoid any kind of hiccups where Alice lacks sufficient resources to fulfill each action. Since all RAM costs are covered by the contract, only CPU and Bandwidth costs remain for the initial actions that remain for Alice.

Table 4 EOS Allocation For Users/Creators

Action	CPU Cost (μs)	Bandwidth Cost (bytes)	Quantity
Open Channel	467	120	1
Declare	530	168	4
Fulfill	680	152	4
Subscribe	1300	218	5
Vote	350	128	5
Transfer Credit	1210	218	1

CPU EOS (ms) cost = $\sim .0025$ [49]

Bandwidth (KiB) cost = $\sim .0004$ [49]

Total CPU (μ s) cost = $467 + 530(4) + 680(4) + 1300(5) + 350(5) + 1210 = 14,676$

Total Bandwidth (bytes) cost = $120 + 168(4) + 152(4) + 218(5) + 128(5) + 218 = 3,348$

Total needed EOS for CPU = $.0025/1000 * 14676 = .0367$

Total needed EOS for bandwidth = $.0004/1000 * 3348 = .0013$

Based on the data in the table above, the initial allocations using the EOSVibes account creation service are adequate to cover the actions mentioned in the narrative. However, it is important to recognize that users consume the CPU resources provided by a specific block producer (based on geography), and depending on traffic, may require users to stake more EOS to properly cover this set of transactions. However, because unstaking CPU resources and network bandwidth returns the same amount of EOS that was initially staked, users can stake EOS from their balance if necessary without losing EOS. After all, if someone intends to use Submerged, they will require a balance larger than .6 EOS if they wish to subscribe to other creators. Considering that it is common for Patreon donations to be in the range of \$1-5, we can assume each subscription is 1 or 2 EOS. Therefore, for a new user, the initial purchase of cryptocurrencies can be around 15 EOS, or roughly \$55.40.

Coinbase, the most popular exchange, charges a 1.49% fee when buying and selling crypto-currencies, which is already a higher rate than other exchanges such as Binance and Bitfinix, which charge a maximum of 1%. With conservative estimates, we can conclude that Alice can pay \$53.80 to fund her account with 13.5 EOS and stake enough EOS to cover her transactions. Assuming that she owns no SUBM tokens, we can use the following function to

determine at what point Submerged becomes a more cost-effective option than Patreon if Alice was a creator.

$$\begin{aligned}
 & \text{where } x = \text{amount rasied from subscribers;} \\
 & P_{EOS} = \text{price of EOS;} \\
 & P_{startup} = \text{initial EOS resource allocation;} \\
 & E_{fee} = \text{exchange fee (between EOS and fiat)} \\
 & S_{fee} = \text{Submerged fee} \\
 & \text{Patreon fee} = .1 \\
 & (P_{EOS})(P_{startup})(1 + E_{fee}) + (E_{fee})(x)(1 - S_{fee}) = (.1)(x)
 \end{aligned}$$

Figure 25 Submerged v. Patreon Equation

If we input the most recent data, where the price of EOS is \$3.69, the recommended allocation is .7 EOS, the exchange fee of .015, and the Submerged fee at .02, we discover that a content creator begins saving money after raising roughly \$30.70.

Estimating Deployment Costs

The table below outlines the resource allocations that the deployers of the Submerged contract require in order for the contract to function properly. We can assume that our application has 1000 content creators with 24 projects each and with 50 unique backers per creator, with total population of 50,000 users.

Table 5 EOS Resource Allocation for Deployment with 50,000 users

Resource	Variables	Calculations
RAM	channel struct = 72 bytes poll struct = 56 bytes project struct = 80 bytes	Table RAM = 32 + (size)(struct bytes)(# of tables)

	<p>channel_sub struct = 16 bytes credit struct = 24 bytes user struct = 32 bytes size of WAST file = 98,856 bytes</p>	<p>RAM for code = 10 * size of WAST file</p> <p>channels table == 72,032 polls table = 1,376,000 projects table = 1,952,000 channel subscriptions table = 832,000 credit table = 1,200,032 users table = 1,600,032</p> <p><i>RAM total for code = 988,560</i> <i>RAM total for tables = 6,200,096</i> <i>Total RAM = 7,188,656 bytes</i></p>
CPU	<p>Assumes 80% of content creators have completely fulfilled their obligations. CPU resources regenerate after a day, so the following calculations target the distribution that occurs at the end of the billing cycle.</p> <p>Check if passed (closevoting) = 695 (μs) Pay channels (paychannel) = 4000 (μs) Credit subscribers (creditsubs) = 1030 (μs) per 15 subscribers</p>	<p><i>Total CPU = 1000(695) + 800(4000) + 200(4(1030)) + 4000 = 5,519,000 (μs)</i></p>
Bandwidth	<p>Check if passed (closevoting) = 120 bytes Pay channels (paychannel) = 218 bytes Credit subscribers (every 15) = 120 bytes</p>	<p><i>Total bandwidth = 1000(120) + 1000(218) + 200(4)(120) = 434,000 bytes</i></p>
EOS Costs	<p>RAM costs = ~ 0.0503 EOS per KiB [47] CPU EOS (ms) cost = ~ .0025 EOS Bandwidth (KiB) cost = ~ .0004 EOS</p> <p>RAM costs = 7,188,656/1000 (.0503) = 361.80 EOS = ~1,335 dollars CPU costs = 5,519,000/1000 (.0025) = 13.80 EOS = ~50.91 dollars Bandwidth = 434,000/1000 (.0004) = .1736 EOS = ~ 0.64 dollars</p>	

As expected, the most expensive resource to maintain for the contract is the RAM to maintain user tables. However, it is important to consider that CPU costs, which are measured in

microseconds, can be highly variable depending on block producers and network traffic. When deploying a contract, developers can choose to specify which block producer (of the top 21) they would like to act as the primary node. If this block producer's infrastructure goes down, then the amount of EOS needed to stake the network will need to increase. For this reason, decentralized applications typically partner with a block producer, such as EOS New York, to ensure that such bottlenecks are minimized.

Depending on EOSIO

At the current moment, the user and deployment costs are relatively cheap due to the price of EOS relative to the computational power offered by block producers. However, as the price of EOS will increase, so will the cost of these resources. At its speculative peak, the price of EOS hovered around 20 dollars, meaning that .7 EOS, the amount needed for someone to use Submerged, would be 14 dollars if the price of EOS to CPU/RAM/Bandwidth remained the same, which would reduce the competitiveness of the platform compared to existing services. Calculating CPU costs also involves an amount of guesswork, as the time to process transactions is measured in microseconds. This inexact method of measuring computation, compared to Ethereum's standard unit of gas, makes it difficult to manage the optimal amount of staked EOS.

Issues Regarding User Experience

Although Scatter makes it easier for individuals to use their EOS accounts, depending on Scatter also brings some issues regarding user experience. For example, Scatter displays the name and parameters of every action a user is about to sign. In theory, this is a great method to

inform the user about what information they are passing to the smart contract. However, recent updates have removed the ability to view a smart contract's Ricardian contract, a human-readable explanation of a contract action. Furthermore, due to naming restraints on the EOS network, the name of an action or its parameters may not map perfectly between what the user imagines they are doing and the related contract action. Similarly, when a user is "creating an account" on Submerged, Scatter asks them to "login." These kinds of linguistic inconsistencies are not completely avoidable and could lead to confusion for users who are already intimidated by a "decentralized application."

Security & Privacy

At the current moment, there has not been any intensive security screening of the Submerged contract. The only action open to the public is creating a channel and creating an account. Both of these actions rely on an account's individual CPU resource. However, RAM is still paid for by the contract, meaning that any new additions to the tables would deplete the contract's resources. However, in the event of a DDOS-like attack, each new addition to the contract would need to come from an account, and each account requires .7 EOS to be created and functional in the first place. Similarly, any attacks on the users table would require the malicious agent to subscribe to a channel, expending EOS. Still, this is not adequate protection against a DDOS attack. Potential solutions could involve email confirmation of an identity, where a user must verify their email, and thereafter a server, with Submerged's active key, then activates the user's account in the channels table.

For funds to be stolen from the Submerged contract, the contract's active key or owner key would need to be exposed. These keys could be exposed by inadvertent sharing of permissions or some attack on the server. It could also be possible for a malicious agent to figure out how to credit themselves within the contract's tables. Otherwise, some security flaw in the underlying eosio.token contract would have to be discovered.

All data in the multi-index tables is public. The only real form of privacy that the application offers is that all EOS accounts are pseudonymous, and all the user data is indexed under these pseudonymous accounts. Other social data, such as posts, are intended to be public in the first place, but are stored in a traditional database and therefore cannot be viewed by non-subscribed users.

Chapter 7

Future Work

EOS Account System

One of the clearest ways to increase Submerged's broader appeal is to move away from depending on Scatter wallets. In other words, allowing users to create a conventional account that is tied to a wallet controlled by Submerged. Ideally, a user could interact with the Submerged application using either Scatter or this built-in wallet system, depending on the user's experience and opinions towards "decentralization." This iteration of Submerged focused on integrating with Scatter wallets because early-adopters would most likely be crypto-enthusiasts and because of development constraints. Controlling users' wallets would ostensibly translate to building an exchange, or at least integrating with an existing one like Coinbase (Coinbase is yet to support EOS). Managing the private keys of potentially thousands of accounts, from a server, is a serious endeavor with significant security risks, as recent exchange-hacks have illustrated. However, this level of abstraction seems like a logical step. It may seem unreasonable to the average user to have to learn the difference between an active and owner key, how to stake EOS tokens, and how to sign transactions.

Stable Coins

This current iteration of the Submerged contract relies on the native EOS token. However, considering its price volatility, over the course of a 30-day cycle, the value of the

funds held in escrow could appreciate or depreciate over thirty percent. Thus, it would seem logical to move to a stable coin solution, where value can be expected to be less volatile during this period. As of the time this paper was written, there is still no leading stable coin in the EOS space, and creating a stable coin specific to the Submerged platform is a significant undertaking in engineering and economics.

A Potential (Obstructed) Path to Launch

As a decentralized application, one of the key aspects is community. Therefore, any real launch would require an initial group of users to serve as Foundation members and creators. The most logical method to acquire this initial community would be to invite existing creators to a beta version of the application and offer them favorable token rewards for doing so. These initial users could also help in developing the “Submerged constitution” which would lay down the rules for Foundation appeals.

The initial capital for the SUBM token would need to be raised in an ICO or “token sale,” ideally from a closed pool of investors rather than the public. Then, once a certain amount of capital is raised, and there is some idea of the size of the community, token rewards can be appropriately calculated.

The most significant challenge in potentially launching Submerged is the legal grey zone surrounding cryptocurrencies. In this case, the SUBM token is indeed a security, as it intended to deliver EOS dividends once a quarter. The lack of an existing framework on how to legally implement such financial arrangements in the US through cryptocurrencies has prompted a significant number of EOS blockchain-based companies to register in tax havens, such as the

Cayman Islands [50], Curaçao [51], the US State of Wyoming and the Cook Islands [52].

Although the SUBM token is not necessary to the Submerged model, it can serve as a method to reward early adopters and can be a source of funding for the developers, both of which are key to developing Submerged further.

Chapter 8

Final Remarks

Submerged in Review

Submerged is a decentralized application that offers audiences' greater accountability and transparency from independent content creators while presenting those same creators with a method to take home more of the money they raise. The application makes use of traditional front-end and server technologies while also relying on blockchain technologies to handle deadlines, escrow, payment, voting, and the fulfillment of promises. Based on the evaluations that took place in the rigor cycle, with the current price of EOS and EOS resources, creators, if they switched from Patreon to Submerged, could begin to see savings after raising \$30.70. While these features are made possible by EOSIO, Submerged is simultaneously vulnerable to shifts in the larger ecosystem. If the price of EOS rises and computational power does not scale appropriately, then the cost savings offered by the application may dissipate. Future iterations of Submerged will require action on both community-building and financial fronts, as the SUBM token and the Submerged foundation require more community and investor input to be fully implemented.

Implications

The most important lesson offered by Submerged is to *not* decentralize “everything.” Indeed, this is one of the principles that allows for EOSIO to be much more scalable than other

infrastructures, and why it is the most “developer friendly.” Although decentralization may be attractive in an ideological sense, requiring so much involvement from users is counterintuitive if a user is simply looking for a service. This may be different when users are looking for a community. Contracts should be made as lean as possible, and only support the key features that a developer wishes to decentralize. In the case of Submerged, these features all surrounded payment and fund distribution. As demonstrated in the evaluation, keeping tables small minimizes the amount of RAM consumed by the contract. RAM, unlike CPU and network bandwidth, is variable to market fluctuations and does not necessarily return the initial amount staked. Minimizing the amount of RAM a contract uses, therefore, exposes developers to less market volatility. With potentially thousands of EOS staked to support a contract, a fluctuation of 5% is significant. Developers also need to consider who pays for the contract’s tables. Requiring individual users to pay for their own tables is a great way to offload expenses, but is really only viable if each table is only used by the table owner. In cases where multiple parties are editing the same tables, then, from a reliability standpoint, it is better for developers to take on these costs themselves. However, doing so can expose developers to malicious agents who wish to spam the contract and reduce the contract’s available RAM. Developers, on a similar note, can make use of deferred actions (paid for by them) to minimize the amount of CPU resources a user needs to interact with their contract. As much as developing a smart contract is “doable,” allowing users to interface with it directly, such as through the Scatter wallet, can lead to confusion or frustration. Not only do such wallets not share the same terminology as the application, but they can expose users to technical details that may overwhelm them. If possible, developers should consider creating their own abstracted interfaces instead of relying on third-

party wallets. Even better, they should aim to create experiences that are so familiar users are unaware the underlying service makes use of a blockchain.

Appendix A

Links to GitHub Repo and Relevant Documentation/Tools

Submerged Repository

The attached repository contains all three components of the Submerged application: contract, front-end, and server. The READMEs include instructions on how to start the application locally.

- <https://github.com/PChwistek/thesis>

EOSIO Repository

The first link refers to the GitHub of EOSIO, where there are links to the relevant EOS APIs and the EOS Contract Development Kit (eosio.cdt) that is responsible for compiling smart contract. The other two links refer to EOS resource tools.

- <https://github.com/EOSIO>
- <https://www.eoscharge.io/>
- <https://www.eosrp.io/>

Scatter Documentation

Scatter provides friendly instructions on how to link apps to Scatter wallet. Conveniently, the ScatterJS APIs is compatible with all Scatter wallets (desktop, mobile, extension, etc.).

- <https://get-scatteer.com/docs/getting-started>

NestJS Documentation

NestJS builds on top of Express, a server framework for Node.js. It provides a very structured way of writing servers with support for static typing, and is a great framework for beginners.

- <https://docs.nestjs.com/>

ReactJS Documentation

React has become an incredibly popular framework for web development in recent years. I've also included links to React Router and React Semantic UI libraries, which are both used in this project.

- <https://reactjs.org/docs/getting-started.html>
- <https://reacttraining.com/react-router/core/guides/philosophy>
- <https://react.semantic-ui.com/>

Redux Documentation

React is primarily meant for managing views. Redux helps us manage an applications state, and allows us to connect each view to the application's state. Redux Thunk helps us manage asynchronous changes to state (API calls).

- <https://redux.js.org/introduction/getting-started>
- <https://github.com/reduxjs/redux-thunk>

Appendix B

Learning Resources

EOSIO

The best way to learn about EOSIO is to go through the documentation on the EOSIO site. The versioning can be a little confusing, but its still better than any other resource I've viewed nonetheless. For development help, I've found the community at StackExchange to be very helpful. EOS New York and EOS Canada provide good learning resources for understanding the broader concepts of EOS.

- <https://developers.eos.io/eosio-home/docs>
- <https://eosio.stackexchange.com/>
- <https://medium.com/eos-new-york>
- <https://www.eoscanada.com/>

JavaScript Development

Since ECMAScript 6 entered the JavaScript ecosystem, it has made JavaScript a much more predictable and developer-friendly language. I highly recommend Tyler McGinnis's (core contributor to React) online courses, as well as a Coursera course offered by The Hong Kong University of Science and Technology as an introduction to JavaScript development. The book, "Secrets of the JavaScript Ninja," is a strong follow-up once one acquires intermediate knowledge of JavaScript.

- <https://tylermcginnis.com/courses/>

- <https://www.coursera.org/specializations/full-stack-react>
- <https://www.amazon.com/Secrets-JavaScript-Ninja-John-Resig/dp/193398869X>

BIBLIOGRAPHY

- [1] Gartner, "Gartner Research Methodologies," 2019. [Online]. Available: <https://www.gartner.com/en/research/methodologies/gartner-hype-cycle>. [Accessed 25 March 2019].
- [2] R. Brown, "Cryptocurrencies have shed almost \$700 billion since January peak," *CNBC*, 23 November 2018.
- [3] J. Kemp, "Wikimedia Commons," 27 December 2007. [Online]. Available: https://commons.wikimedia.org/wiki/File:Gartner_Hype_Cycle.svg. [Accessed 25 3 2019].
- [4] G. F. Knapp, *The State Theory of Money*.
- [5] S. Haber and S. Stornetta, "How to time-stamp a digital document," *Journal of Cryptology*, 1991.
- [6] A. Narayanan, J. Bonneau, E. Felten, A. Miller and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies*, Woodstock: Princeton University Press, 2016.
- [7] The Economist, "The great chain of being sure about things," *The Economist*, 31 October 2015.
- [8] Azaghal, "Wikimedia Commons," 25 January 2012. [Online]. Available: https://commons.wikimedia.org/wiki/File:Hash_Tree.svg. [Accessed 25 March 2019].
- [9] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.
- [10] A. M. Antonopoulos, *Mastering Bitcoin: Programming the Open Blockchain*, Sebastopol: O'Reilly Media, Inc., 2017.
- [11] The Economist, "Why bitcoin uses so much energy," *The Economist*, 9 July 2018.
- [12] Bit Info Charts, "Bitcoin Avg. Transaction Fee historical chart," 24 1 2019. [Online]. Available: <https://bitinfocharts.com/comparison/bitcoin-transactionfees.html>. [Accessed 24 1 2019].
- [13] I. Takashima, *Ethereum: The Ultimate Guide to the World of Ethereum*, Amazon, 2017.

- [14] N. Szabo, "Smart Contracts: Building Blocks for Digital Markets," 1996. [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html. [Accessed 24 1 2019].
- [15] Delphi, "The Oracle Problem," *Medium*, 15 July 2017.
- [16] J. Peterson, J. Krug, M. Zoltu, A. K. Williams and S. Alexander, "Augur: a Decentralized Oracle and Prediction Market Platform," Forcast Foundation, 2018.
- [17] Ethereum Community, "Ethereum Docs," 2016. [Online]. Available: <http://ethdocs.org/en/latest/introduction/what-is-ethereum.html>. [Accessed 24 January 2018].
- [18] G. Wood, "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER," 2014.
- [19] J. Frankenfield, "Crypto Token," *Investopedia*, 3 April 2018.
- [20] J. Biggs, "How to run a token sale," *TechCrunch*, 2017.
- [21] V. Gupta, "A Brief History of Blockchain," *Harvard Business Review*, 28 February 2017.
- [22] J. Liebkind, "Is EOS the New BTC? Pay Attention to Peter Thiel," *Investopedia*, 9 August 2018.
- [23] J. Kauffman, "What is the Role of a Block Producer," *EOSCanada Blog*, 23 April 2018.
- [24] J. Kauffman, "How Do I Use My EOS Tokens to Vote for a Block Producer?," *EOSCanada Blog*, 24 March 2018.
- [25] J. Kauffman, "What Does Staking and Unstaking EOS Tokens Mean?," *EOSCanada Blog*, 22 August 2018.
- [26] EOSIO, "EOSIO.CDT (Contract Development Toolkit)," 15 January 2018. [Online]. Available: <https://github.com/EOSIO/eosio.cdt>. [Accessed 25 January 2018].
- [27] P. Bright, "The Web is getting its bytecode: WebAssembly," *Ars Technica*, 18 June 2015.

- [28] EOSIO, "EOSIO Developer Portal - Upgrading the System Contract," 2019. [Online]. Available: <https://developers.eos.io/eosio-cpp/docs/upgrading-the-system-contract>. [Accessed 25 January 2019].
- [29] EOSIO, "EOSIO Developer Portal - Communication Model," 2019. [Online]. Available: <https://developers.eos.io/eosio-cpp/v1.3.1/docs/communication-model>. [Accessed 25 January 2019].
- [30] EOSIO, "EOSIO Developer Portal - Multi-Index DB API," 2019. [Online]. Available: <https://developers.eos.io/eosio-cpp/v1.3.1/docs/db-api>. [Accessed 26 January 2019].
- [31] eostoolkit, "What are Active and Owner Keys and Permissions?," *EOS Help Desk*, 26 June 2018.
- [32] R. Cannard, "Worst day of my Crypto Life.....never thought I would be scammed," Reddit, 26 January 2019. [Online]. Available: https://www.reddit.com/r/eos/comments/ajzsr/worst_day_of_my_crypto_lifenever_thought_i_would/. [Accessed 26 January 2019].
- [33] EOSLynx, [Online]. Available: <https://eoslynx.com/>.
- [34] S. O'Neal, "EOS Proves Yet Again That Decentralization Is Not Its Priority," *Cointelegraph*, 15 November 2018.
- [35] Finder, "'Shock' allegations of EOS block producer collusion shock no one," 2018. [Online]. Available: <https://www.finder.com.au/shock-allegations-of-eos-block-producer-collusion-shock-no-one>. [Accessed 26 January 2019].
- [36] Block.one, "Oct 1, 2018 Statement: EOS Public Blockchain Governance," *Block.one Blog*, 1 October 2018.
- [37] K. Kelleher, "Civil, a Blockchain-Media Startup, Cancels Its ICO, Offering a Full Refund to Those Who Bought Tokens," *Fortune*, 17 October 2018.
- [38] Civil, "Civil: Self-Sustaining Journalism," *Civil Blog*, 11 July 2017.
- [39] A. R. Hevner, "A Three Cycle of Design Science Research," *Scandinavian Journal of Information*

Systems, 2007.

[40] P. Martinez, "Surviving the Adpocalypse: Why YouTube Creators Embrace Merch," 26 June 2018.

[41] J. Alexander, "The Yellow \$: a comprehensive history of demonetization and YouTube's war with creators," *Polygon*, 10 May 2018.

[42] O. Seitz, "Patreon Blog," Patreon, 28 June 2018. [Online]. Available: <https://blog.patreon.com/6-membership-based-business-models-you-can-use-on-patreon-today>. [Accessed 16 1 2019].

[43] "Patreon," 28 June 2018. [Online]. Available: <https://www.patreon.com/>. [Accessed 16 January 2019].

[44] "Channel memberships eligibility, policies, & guidelines," YouTube, 16 January 2019. [Online]. Available: https://support.google.com/youtube/answer/7636690?hl=en&ref_topic=9153998. [Accessed 16 January 2019].

[45] DappRadar, "Dapp Rankings," 17 February 2019. [Online]. Available: <https://dappradar.com/rankings>. [Accessed 17 February 2019].

[46] Reichheld, Fred; Markey, Rob; Bain & Company, *How Net Promoter Companies Thrive in a Customer-Driven World*, Boston: Harvard Business Review Press, 2011.

[47] D. Larimer, "Developing Efficient Contracts," 12 December 2018. [Online]. Available: <https://medium.com/@bytemaster/developing-efficient-contracts-8a8e62011c6d>. [Accessed 25 March 2019].

[48] "EOS Charge," EOS New York, 25 March 2019. [Online]. Available: <https://www.eoscharge.io/>. [Accessed 25 March 2019].

[49] "EOS Resource Planner," EOS New York, 25 March 2019. [Online]. Available: <https://www.eosrp.io/>. [Accessed 25 March 2019].

[50] "Crunchbase," Crunchbase Inc., 25 March 2019. [Online]. Available:

<https://www.crunchbase.com/organization/block-one>. [Accessed 24 March 2019].

[51] EOSBet Casino, "EOSBet Gets Ready for Mainstream Adoption with Account System Launch," *Medium*, 4 January 2019.

[52] EOS New York, "EOS New York: Ownership Disclosure & Corporate Structure," January 2018. [Online]. Available: <https://steemit.com/eos/@eosnewyork/eos-new-york-ownership-disclosure-and-corporate-structure>. [Accessed 25 March 2019].

[53] D. Gerard, *Attack of the 50 Foot Blockchain: Bitcoin, Blockchain, Ethereum & Smart Contracts*, Amazon, 2017.

[54] M. Iansiti and K. R. Lakhani, "The Truth About Blockchain," *Harvard Business Review*, February 2017.

[55] A. Breen, "How and Why Developing for Ethereum Sucks," *Medium*, 19 January 2018.

[56] D. Sui, J. Pfeffer, J. Gillis and E. Muzzy, "A Retrospective of the EOS Token Sale," *Consensus Media*, 25 October 2018.

[57] M. Iles, "The Civil White Paper," *Civil Blog*, 11 May 2018.

[58] EOSBet Casino, "EOSBet Update: Dividends!," *Medium*, 29 September 2018.

Philip Chwistek - Academic Vita

Education

Penn State University, Schreyer Honors College B.S. Information Sciences and Technology B.A. English	University Park, PA August 2015 - Present
Maastricht University Study abroad concentrated on international business	Maastricht, Netherlands July 2016 - August 2016

Work Experience

Zuper Superannuation Junior Developer (Remote)	Sydney, Australia September 2018 – Present
<ul style="list-style-type: none">Coordinated with design team to create a jQuery lookup widget compatible with InstapageImproved load times (3x) of client side rendered tree mapsCreated a web-based personality/financial profiling tool with a behavioral psychologist	
Software Development Intern Sage Corps Fellow	July 2018 – August 2018
<ul style="list-style-type: none">Created responsive React/Sass components for Contentful (CMS) modelsImplemented UI improvements for onboarding process	

ServiceDock Full-Stack Development Intern Sage Corps Fellow	Dublin, Ireland July 2017 – August 2017
<ul style="list-style-type: none">Designed and implemented a web-based data visualization solution for customer feedback<ul style="list-style-type: none">CEO demoed functionality to franchises, such as Centra (convenience store chain)	

Center For American Literary Studies Undergraduate Research Assistant	University Park, PA August 2017 – May 2018
<ul style="list-style-type: none">Wrote a book review that aired on local NPR station, aided in planning of graduate symposia	

College of IST Learning Assistant (Introduction to Programming)	University Park, PA August – December 2016
<ul style="list-style-type: none">Assisted instructor with creating assessments, performed 1-1 instructionGraded student tests and problem sets	

College of IST Undergraduate Research Assistant	University Park, PA April – August 2016
<ul style="list-style-type: none">Improved a recommender system prototype funded by a FBI grant	

Fox Chase Cancer Center Bioinformatics Intern	Philadelphia, PA June – August 2014
<ul style="list-style-type: none">Wrote Python scripts to analyze human DNA and RNA using BioPython	

Skills and Technologies

- Languages/technologies: Proficient in Java, JavaScript; Familiar with Python; previously used React, Node.js, Spring MVC, Solidity, Sass, C++. Experienced with Git, Sketch, and UML.
- Strong analytical and creative writing, fluent in Polish

Publications

- Bagby, John W., David Reitter & Philip Chwistek, *An Emerging Political Economy Of The Blockchain: Enhancing Regulatory Opportunities*, 88 UMKC L.Rev. 1-54 (Sept.2019)

Independent Projects

Clairvoyance

Full-Stack Developer

Warrington, PA

December 2017 - Present

- Developed a dapp for users to bet Ether on eSports games (React, Node.js, Solidity)

Awards, Grants, and Honors

- Edward M. Frymoyer Honors Scholarship 2018
- IST Honors Scholarship 2018
- Ann Goode Moore and Howard R. Moore Jr. Scholarship in English 2018
- Schreyer Travel and Research Grant (\$2500) 2018
- College of Liberal Arts Enrichment Grant (\$1500) 2018
- College of Liberal Arts Enrichment Grant (\$1850) 2017
- Penn State Student Engagement Network Grant (\$2000) 2017
- Schreyer Travel and Research Grant (\$1250) 2017
- MC for the IST Donor Dinner 2016
- National AP Scholar 2015
- Frederick Douglass and Susan B. Anthony Award for Social Understanding 2014