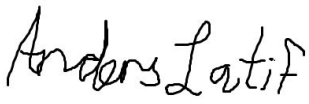
Democracy manifest

ethereum

Anders Latif - 21/08/1988



KEA - Datamatiker
DAT15A - 5. Semester
Afslutnings Opgave
03/11 - 2017

1 TABLE OF CONTENTS

2 INTRODUCTION

Notice: there is a glossary list. If a word is used for the first time it will be in italics.

2.1 Synopsis

My interest in cryptocurrency was piqued last semester where I played around with Bitcoin by creating my own altcoin clone. I fully understood that blockchain technologies are emerging. One of the most ambitious projects on that front is brought forward by the Ethereum Foundation.

The Ethereum blockchain contains the protocol for clients to run the Turing complete Ethereum Virtual Machine which can hold state and interpret code. Contracts can be deployed to the blockchain and run from all nodes in the network. My goal is to deploy smart contracts that will contain code for holding open and closed type of elections. I have chosen to write my contracts in Solidity and I will go into depth about quirks in this new language.

With my decentralised application (Đapp) I will then create a voting platform accessible through the current internet. The website will be built with React. I will investigate how I can have a Node.js server call and transact to the contracts.

As this is a solo assignment I will not focus on project management. As the end-goal is open-source code I will also sidestep discussions about organisational theory. My thesis is mainly focused on the code I produce and the design patterns. The interesting thing is that design patterns for the blockchain are very different than traditional ones and I will take a deep look into those fundamental differences.

Just like in the beginning of the thesis where I will show how Ethereum is evolved compared to Bitcoin I will bring it to a full circle in the end by reflecting on where blockchain development is headed in the future. I will bring up current criticisms to showcase the problems that need to be solved.

I believe that creating a democracy on the blockchain is an apt example to illustrate the social impact of having verifiable code. We will explore the main question of how an open blockchain can help in creating a voting platform that is trustworthy and reliable.

2.2 Introduction to the blockchain

An introduction to blockchain technology starts with Bitcoin. Books can and should be dedicated to it. It's hard to describe what Bitcoin is because it is actually several things[1]. Broadly, it can be described as a consensus protocol. It is the revolutionary solution to the *Two Generals' Problem* / *Byzantine Generals Problem* / *Byzantine Fault Tolerance* (BFT). The problem can be stated with network terms like this: how can we ensure trust through a *P2P* (peer to peer) network using the BitTorrent Protocol? Notice that the aim is a decentralized solution since all peer nodes are equal (thus the name).

Satoshi Nakamoto laid out his proposal for solving BFT in his White Paper. The idea is to have all miners validate the transactions in a block. But that means that anyone can attack the blockchain? Indeed it does. But Satoshi introduces the concept of *Proof of Work*. In order to mine a block the miner has to go solve a cryptographic puzzle.

It takes on average 10 minutes to solve the puzzle. More nodes in a network competing to solve the puzzle won't mean that the puzzle will be solved faster. The election cycle will still average on 10 minutes by adjusting the difficulty of the puzzle.

A serious mining rig consists of chips called ASICs that are built to solve the puzzle. So as you can imagine verifying a block is heavy in resources (electricity, hardware cost, hardware tear). Herein lies the work in the concept Proof of Work. By willingly sacrificing "work" you prove that you care to uphold the protocol.

Now an attacker has to consider whether it's worth it to invest all the work into being evil. They can choose to be evil "just because" and lose money on it or they can choose to be good and be rewarded with an incentive: Bitcoins. Most choose the latter and that is why the algorithm works because it would also require more than 70% of the nodes in the network to overthrow the protocol.

The incentive serves two purposes:

1. It pays good miners and keeps them honest
2. It introduces new bitcoins to the market (hence the name miner)

This is the essence of Proof of Work. A consensus based trust-model is achieved by creating a blockchain token and incentivizing nodes.

[1] It won't be mentioned further but it is also a distributed database (of the blockchain itself), a network of miners, routing nodes or full nodes and then a stateful application layer (wallets).

2.3 How ethereum differs from bitcoin

Proof of work is both elegant and problematic. Besides the obvious wastefulness the result at this growing rate is that soon enough only supercomputers will be able to mine. Long gone are the days where personal computers can mine and we are already seeing the shift from personal mining rigs to dedicated server farms mainly in China[2].

Two alternatives have been proposed by Ethereum. *Proof of Stake* temporarily freezes up assets and rewards nodes who have a lot of stake in serving the protocol. Proof of Excellence is a completely new protocol that finds nodes that are in congruence with other nodes and rewards them for it. Neither of these models require excessive CPU usage.

The limitation with bitcoin is that it's simply a ledger and nothing more. Each block contains rows of transaction information: amount, sender and receiver. Knowing this it is obvious that Bitcoin is stateless. So if I want to spend Bitcoin how do I know if I can spend them? I would have to traverse the entire *Merkle Tree*[3] (blockchain) and see when I received coins, when I spend them and my balance would be the accumulative sum. The blockchain is growing at an alarming rate and having to traverse it is impractical for most users. Ethereum on the other hand allows for state to exist in blocks. In fact, Ethereum provides so much more than a public ledger.

3 ETHEREUM DEVELOPMENT

3.1 Evm

It is indeed the case that Bitcoin is nothing more than an attempt to create a decentralized fiat valuta. There have been scripting attempts on the Bitcoin blockchain such as *Namecoin* or *UTXO* (unspent transaction outputs) but it's obvious that the result is very simplistic since this is not what Bitcoin was originally intended for. Ethereum on the other hand is created with blockchain development in mind. It has protocol specifications for accepting bytecode so each client on top of it comes strapped with *Ethereum Virtual Machine* (EVM).

[2] Top 10 Mining Pools in 2017: https://bitcoinworldwide.com/mining/pools/

[3] A Merkle Tree is the cryptographic terminology for a binary hash tree. The goal of the blockchain is to link blocks with hashes so there is only one child per parent. But while the blocks are added to the blockchain and verified they might appear as Orphan or Uncle blocks. The bitcoin protocol corrects this.

The EVM is Turing-complete and stateful. It allows for several languages to run on iit as long as they compile to the specifications. Each *contract* (similar to an object in OOP) can hold data in 3 different places (permanently, semi-permanently or temporarily).

3.1.1 Memory

When talking about the Ethereum blockchain being stateful and being able to run code on the EVM the most pressing question becomes how does it store memory? Ethereum is able to store state in 3 different kinds of memory. The 3 memory types in order of longevity (longest first)[4]:

1. Storage (Indefinite part of the object)
2. Memory (Within the scope of the transaction/call)
3. Stack (Until stack is popped)

First place data can be stored is the "storage" where all the contract state variables reside. Every contract has its own storage and it is persistent between function calls and quite expensive to use.

The second is “memory” defined by the keyword of same name, this is used to hold temporary values during a transaction and immediately released after the transaction. The data in memory is erased between (external) function calls. An example from my own project where I define two variables that only need to exist in the context of the method as memory:

```
function getResults() constant returns (bytes32[], uint[]) {

       bytes32[] memory proposalNames = new bytes32[](proposals.length);
       uint[] memory counts = new uint[](proposals.length);

       [...]

      return (proposalNames, counts);
}
```

The example has an interesting feature which I have only seen in Solidity. The return values are enshrined with parenthesis which is called a pipe and it returns the values as an array of 2 indices.

---

4 https://solidity.readthedocs.io/en/latest/frequently-asked-questions.html#what-is-the-memory-keyword-what-does-it-do

The third memory location is the stack, which is used to hold small local variables. It is almost free to use, but can only hold a limited amount of values. The stack is maintained completely by EVM and the user can't do anything to change its behavior.

The aim is to move as much as possible down the ladder to the smallest scope. This is because storing data is much more costly than computational cost which makes sense cause storage memory on the blockchain needs to be stored on all computers on the network. And changed everywhere after being changed somewhere once.

3.1.2 Variable sizing

In the code example in previous sub-chapter I conveniently skipped past explaining the variable types: byte32 and uint. Both names relate to the size of the variable type.

Uint stands for unsigned int. In Solidity it's possible to define an integer as either int or uint. In many cases it is known in advance that negative values do not make sense (amount of people, a wallet address) so Solidity allows you to declare the integer as unsigned to get a higher positive range. It's also possible to declare the integer size from 8 up to 256 with 8 bit increments[5]. In this case I haven't defined a size so it defaults to uint256.

Knowing all this about uint *byte32[ ]* makes a lot of sense all of a sudden. It's a declaration of an array of 32 byte strings.

3.1.3 Gas

"*The currency is in the service of the protocol*"[6] - Vitalik Buterin in relation to Gas and how it's the opposite for Ethereum from Bitcoin.

We have talked about the costliness of operations and how to limit space usage which is nice on the nodes in the network. But what ensures that every contract developer is nice? What prevents a rogue developer from writing an infinite loop and taking down the network indefinitely?

[5] http://solidity.readthedocs.io/en/develop/types.html#integers
[6] Decentralizing Everything with Ethereum's Vitalik Buterin | Disrupt SF 2017: https://youtu.be/WSN5BaCzsbo?t=20m26s

These are the questions that Vitalik Buterin has solved. He introduced a new principle within the crypto-world which is Gas and lays out the foundation in the Ethereum White Paper[7] where he calls it "*Ethereum's anti-denial-of-service model*"[8]. Gas is an amount of cost withdrawn when performing computations.

EVM has a set of rules to calculate the gas cost of transacting to a contract. The estimated gas cost is based on expected computational cycles. Gas is a small unit transferred from Ethereum's own coin.

The conversion from Eth coin to gas varies. First of all you can define the gas price yourself over a minimum. It makes sense that the higher the gas price you set the faster mimers will choose your transaction to mine. The minimum is set by the Ethereum foundation and the lower bar was raised this summer of 2017 to some controversy and market instability. Gas is a topic that Ethereum investors speculate a lot about.

One thing that needs to be mentioned is that gas can't be owned, only spent. It is converted from Eth coin from a wallet when needed. In short, deploying a contract or changing state isn't free.

Gas makes it interesting for a developer to create for the blockchain. It requires smart developers[9] and values efficiency in the code because the smallest surplus of computational cost can get blown up in the network.

One very obvious effect that gas has on the coding is that *for* and *while* loops are avoided since each loop cycle is costly. Using a *mapping* (like hashmap) is to be prefered.

Just to get an idea how much gas a normal contract costs to deploy please look at figure 3 later on where I showcase a contract compiled in Remix and the calculated estimated gas cost for deployment is included.

3.2 Networks

Just like the Bitcoin having a main network and 2 test networks Ethereum has several versions of the main network and many test networks.

[7] https://github.com/ethereum/wiki/wiki/White-Paper
[8] https://github.com/ethereum/wiki/wiki/White-Paper#messages-and-transactions
[9] Personal Anecdote: In the listings in the private Ethereum foras I frequent I often encounter huge money bounties from ICO's for code bugs/improvements.

Test networks exist as separate blockchains different from the main network. The purpose is to enable developers to experiment with their code before deploying it to the main network.

While this significantly reduces the waiting time the test networks still work the same way and require miners to mine the transaction in a block. That isn't opportune so simulations exist. I use *TestRPC* which simulates an ethereum blockchain locally with 9 different wallets containing Eth coins. These coins can be converted to gas and transactions are mined instantly without waiting time.

Alternatives for the command line interface TestRPC there is a Go version called *Geth*. Also, while writing this very sentence in mid-November it has just been announced that Truffle will implement their own test blockchain simulation built on Ganache Core which comes from TestRPC.

There is also a third option. Of course it's also possible to host your own Ethereum blockchain clone where the *genesis block* is mined from scratch by yourself[10].

Below is an overview of the different Ethereum networks. There are different code names for the blockchain that each indicate a major change in the blockchain. Each change in the main network is still on the same blockchain but is like a snapshot of the blockchain at a certain point of time. In short, they still share the same genesis block.

	Main network
	Test network

Version	Network Id	Code name	Release date	Notes
0	0	Olympic	May, 2015	Referred to as a testnet because it was pre 1.0 release
1	1 [chain ID: 61]	Frontier	July 30th, 2015	Sometimes called Ethereum Classic
2 [alternatively 1.1]	1	Homestead	March 2016	Upgrade from Frontier

[10] The genesis block is the first block in the blockchain that has to be mined in a specific way (not by nodes in the network). It is the merkle tree root.

1	2	Morden	November 20th, 2016	
2	3	Ropsten	February 2016	Attacked and revived in March 2017
	42	Kovan		Introduced after the Ropsten attacks. Introduces proof-of-authority
3 [Referred to as Ethereum 1.2]	1	Metropolis (vByzantium)	October 16th, 2017 [Current]	Introduces decentralized application browsers and ĐApp store
	4	Rinkeby	TBA	PoA and clique consensus
3.5		Metropolis (vConstantinople)	TBA	
4 [Referred to as Ethereum 1.5]		Serenity	TBA	

Figure 1, 2: The past, current and yet to come version names of the Ethereum networks color coded for main and test networks. The numbering and order can be confusing, so hopefully this helps. The information is scattered around the internet and not all available which explains the holes.[11] [12] [13].

After studying the beautiful table one question remains. How are changes enforced? Doesn't that go against the very idea of the currently employed *PoW* algorithm which is placed to ensure that the blockchain can't be manipulated or taken over?

The changes are enforced through forks either by users or miners. They require nodes to agree to a change and follow the same path on the split.

In blockchain technology there is a distinction between soft and hard forks. Soft forks restrict the rules for block access while a hard fork is a breaking change that isn't

[11] https://en.wikipedia.org/wiki/Ethereum#Milestones

[12] https://ethereum.stackexchange.com/questions/10311/what-is-olympic-frontier-morden-homestead-and-ropsten-ethereum-blockchain

[13] https://testnet.etherscan.io/

forward compatible[14]. A hard fork is thus a major change to the Ethereum protocol that requires all nodes to adhere to it[15].

Usually the agreed upon branch is continued with but sometimes the community is split and choose to continue separately. It happened in one case for Ethereum and now there is a split in the blockchain and therefore two coins exist: Ethereum Classic (ETC) and Ether (ETH). It happened because of *The DAO* attack where $50 million dollars was lost[16]. Though The DAO is an entity outside of Ethereum it was felt that it would scrutinize the entirety of Ethereum in the public eye and it was agreed upon by the majority to go back to before the hack happened. A group of code purists who hold immutability above personal concerns disagreed and continued with the original branch.

If we can learn one thing from The DAO hack it is to be absolutely sure that the contract does not contain an exploit. It matters that smart contracts are smart.

4 CHOICES MADE

4.1 Choosing a high-level language

Six programming languages have been created for Ethereum. One is deprecated, others are poorly maintained. The two strongest candidates are Solidity and Go-Ethereum.

Solidity is like a mix of Javascript and Java. Though the two languages are very far from each other the comparison is apt because it is like an object oriented typesafe version of Javascript. Solidity is by far the most popular language for Ethereum contract development and has a good documentation.

Go-Ethereum is, as the name implies, the Golang implementation of an Ethereum contract language. It is not as well-documented and maintained as Solidity.

This very day in the middle of November a new language called Viper has been announced and released. Viper is an experimental language that is similar to python. It is being called experimental since no documentation has been written for it yet. It boasts of type safety and focus on readability rather than ease of writing the code.

[14] https://bitcoin.stackexchange.com/questions/30817/what-is-a-soft-fork
[15] https://www.investopedia.com/terms/h/hard-fork.asp
[16] https://www.coindesk.com/understanding-dao-hack-journalists/

These two points mean that it lacks some otherwise nice features such as modifiers [17] and class inheritance.

Despite the varying features all the languages compile to the same bytecode and it is mainly a matter of syntactic preference when choosing one instead of another.

I've made the design choice to code my project in Solidity. Solidity is a fitting choice considering its proximity to Javascript which fits with my choice to create a website in React with Node.js. NPM mostly only has packages for contract development in Solidity.

Another strong argument for Solidity is that it "*supports inheritance, libraries and complex user-defined types among other features.*"[18] This pairs well with the relative complexity of my application and the deployment system in Truffle. I will be actively using these complex types in my application.

4.2 Solidity

All solidity files end with the extension .sol. In order to compile the code into bytecode that EVM can process the solc compiler is required. At the beginning of every solidity file the first line defines the compiler version like this:

```
pragma solidity ^0.4.2;
```

The pragma keyword comes from C and means that the compiler version 0.4.2 is requested.

Depending on the object type (struct, library, contract) one of the three keywords are used to define an object. It's possible to have several objects in one file.

To showcase one smart contract written in Solidity I have a very simple token example written in an online (browser-based) Ethereum compiler called Remix. The smart thing about the compiler is that it can compile your contract into the bytecode that you can deploy through Javascript without ever installing "ethereum related libraries" or an IDE plugin.

[17] I have a paragraph about modifiers later and it's also defined in the glossary.
[18] https://solidity.readthedocs.io/en/develop/

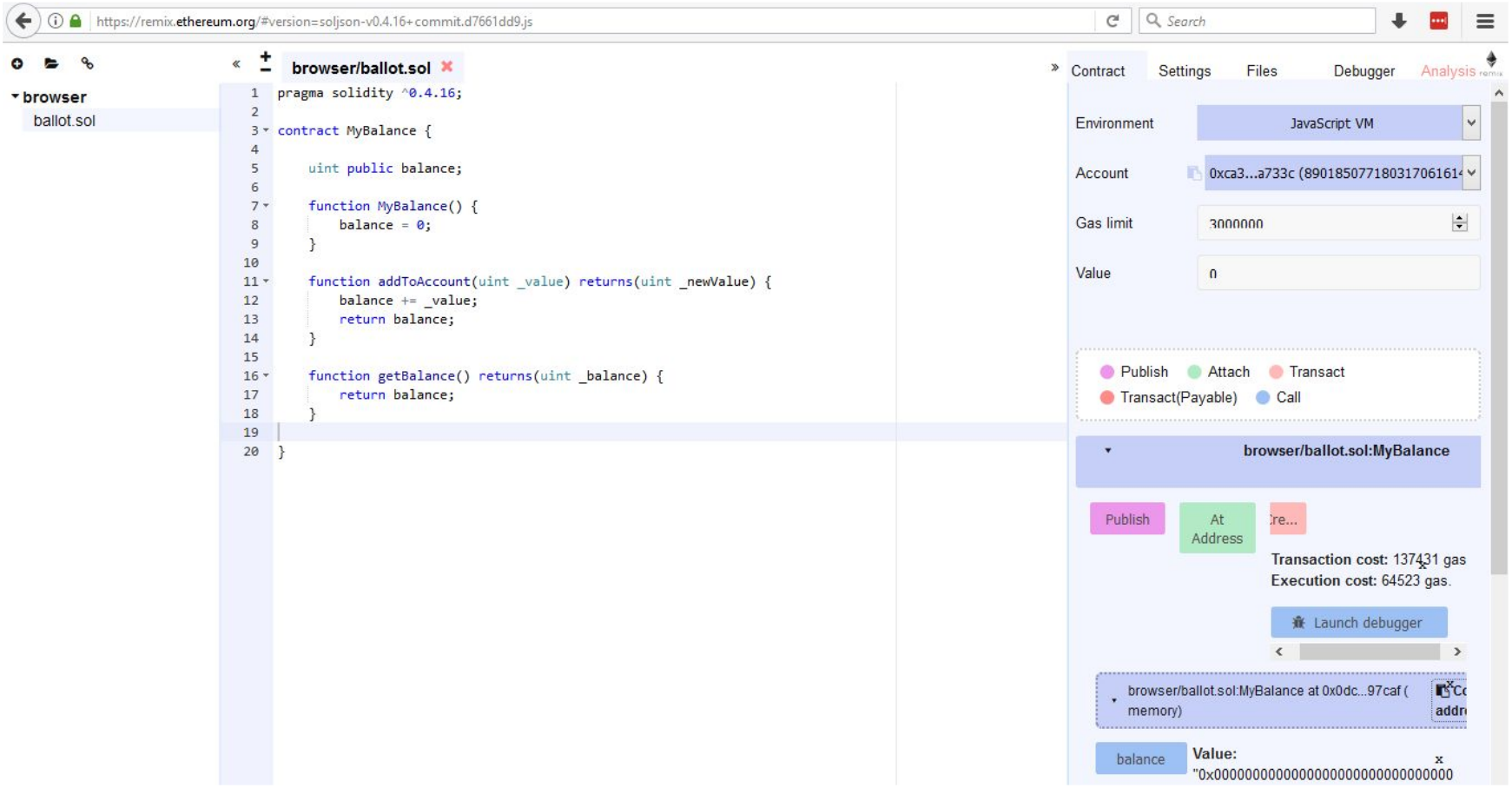

https://remix.ethereum.org/#version=soljson-v0.4.16+commit.d7661dd9.js
Search
browser
ballot.sol
browser/ballot.sol
pragma solidity ^0.4.16;
contract MyBalance {
uint public balance;
function MyBalance() {
balance = 0;
}
function addToAccount(uint _value) returns(uint _newValue) {
balance += _value;
return balance;
}
function getBalance() returns(uint _balance) {
return balance;
}
}
Contract
Settings
Files
Debugger
Analysis
Environment
JavaScript VM
Account
0xca3...a733c (8901850771803170616
Gas limit
3000000
Value
0
Publish
Attach
Transact
Transact(Payable)
Call
browser/ballot.sol:MyBalance
Publish
At Address
Transaction cost: 137431 gas
Execution cost: 64523 gas.
Launch debugger
browser/ballot.sol:MyBalance at 0x0dc...97caf (memory)
balance
Value:
"0x0000000000000000000000000000000

Figure 3: Remix, the online Ethereum compiler. Notice the current transaction and execution gas cost for a simple contract. Gas is on the right side of the picture, near the lower middle above the "Launch debugger" button.

4.3 The Truffle framework

Everything has a simple beginning and this project is not an exception. I used to compile my contracts manually and interact with them in my node console. After a while I started seeking a framework to help me with automation. A framework that comes bootstrapped with convenient commands for deploying contracts onto the network.

There are 2 popular frameworks. Truffle is the most popular framework for Ethereum deployment.

There is also Embark which at the time of writing could boast of simulation features. But just today while writing this sentence Truffle has announced that their framework will also come with its own test network (based on *TestRPC's Ganache*[19]). So this means that Truffle now scores better in all categories between the two.

Once again (just like for Solidity) I chose the most popular framework. The argument for that is how well documented it is. I value good documentation since I have a lot of sour experiences working with libraries that do not have proper documentation. This is also the reason why I early on ensured that I could create documentation from my own smart contracts.

Let's have a look at how truffle works. After running the command "*truffle init*" a specific folder structure is set up.

All contracts are kept in the folder */contracts*. Calling "truffle compile" in the command line compiles all sol contracts to the */build/contracts* folder as JSON files. In them we find an array called *ABI* (Application Binary Interface) where the ABI definition is placed. The ABI definition contains all data types in the sol contract. As the name suggests it is the code that is accepted by the binary interface in the Ethereum blockchain when creating/running a smart contract.

Truffle keeps track of the recent migrations in a migration contract, *Migrations.sol*, and handles whether it needs to (re)deploy newly changed contracts. This is very much like database migrations.

[19] Will go into it in the network paragraph and is defined in the glossary.

Truffle also has its own console where it's possible to interact with the deployed contracts.

Truffle Box, which now is part of Truffle, comes with a pre-packaged project structure. I have personally worked together with the creator of Truffle Box to debug and correct the errors in the React-Auth App since it wasn't working. We solved the issues and now everyone on the world wide web can use a working template. My project is built on its foundation but with improvements in the syntax and the Redux pattern changed to a more proper structure. See more in the Redux chapter.

Finally, Truffle also boasts a testing build for asserting sol contract values. I will go into detail about Truffle Tests under the chapter Tests near the reflective part of this assignment.

4.4 Web3

I have previously mentioned Truffle console which is one way to interact with deployed contracts. Another solution is Metamask which is a browser plugin/extension that injects the eth wallet (user account) into any site and lets you accept/receive eth transfers. A third solution is Mist which is a standalone browser created for viewing Đapps.

Finally there is Web3. Web3.js is a callable object that can receive the ABI array from your compiled contract and use it to interact with the blockchain. It will also give you the return values as JSON.

I have mentioned ABI before but will now expound upon it. The ABI is a statically typed[20], pre-known[21] (as opposed to known at run-time) defined interface that is essential for interacting with the blockchain either from outside of it or between contracts. This is where the web3 object comes handy. Its role is to provide easy access through the ABI and helper methods that can parse ABI definitions and recreate Javascript contract objects.

There are 2 ways to interact with the blockchain:

1. Transactions
2. Calls

[20] https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI
[21] https://solidity.readthedocs.io/en/develop/abi-spec.html

It's crucial to understand the difference. Transactions changes the state on the blockchain/of the smart contract which requires the miners in the network to mine it in a block.

A call is a query that doesn't affect the state. Even calling a function that does computations within the function scope will be completely "free" since the blockchain does not need to know that you've called the function.

All transactions in the Ethereum blockchain return a Promise. In a real time world wide network it means that the Promise won't return a value until at least when the block has been mined. Promises are not chained functions that cause bottlenecks - which is a problem in Node since it has only 1 main thread - but they are callback functions with nicer syntax. In a real world network of varying amount of nodes it's unpredictable when the transaction is mined.

4.5 Use cases[22]

Now that I am settled on my technology stack I want to go through some propositions for what a Đapp can be created for before settling on Democracy Manifest. Everyone seems to agree that blockchain is going to affect the world, yet people are still in doubt what it can be used for. It is especially hard to argument how an idea is improved through blockchain development.

The use cases can be split into three categories:

1. On platform assets - cryptocurrency
2. Off platform assets - ownership of property in the real world
3. Smart contracts - conditional transaction -> if something in the real world happens then do something

I have already explored cryptocurrencies. Off platform assets being managed on the blockchain will be quite revolutionary. The logistics industry is seeking to implement this technology since shipping containers on average travel between a lot of companies and they believe that blockchains can help them keep track of shipments. It could also be used privately. If I want to buy a car from my neighbor, we could transfer the right to the car through the blockchain and I can be its proven owner.

Next we have another moving field: smart contracts. Smart contracts can replace mediators such as insurance people, lawyers etc. Smart contracts are stateful contracts where a change happens depending on a condition. These conditions

[22] Here is a extensive list of Đapps created for Ethereum: https://www.stateofthedapps.com/

could occur depending on the blockchain itself (datetime change) or external things (api calls etc.).

Major lawyer firms are researching into both smart contracts and using contracts for ownership cause it could replace a lot of judicial bureaucracy. The blockchain could also be used as a substitute to a patent or copyright ownership by submitting your work onto it.

Other use cases include using the blockchain as a storage. One example is for registration of identity. It could also be used for trustworthy point keeping systems which is why a lot of games have been developed on the Ethereum blockchain.

I am personally compelled by the idea of using the blockchain to collect scientific data. Imagine for instance a weather reporting system where you let anyone report the weather from where they are. The protocol will then reward through tokens those who are in congruence with the other nodes in the system who are in their area. Anyone could hook instruments up to a Raspberry Pi and broadcast the data. This could create the biggest weather station network in history and give us the most accurate weather data known to man.

I have chosen to explore the possibility of creating a Decentralized Autonomous Organisation that lets people vote on proposals. This, I believe, illustrates the social impact of having trustworthy code on the blockchain.

5 DEMOCRACY MANIFEST

You've come this far and by now you should've understood all the moving parts to create my example project. Democracy Manifest is decentralized voting platform built on Ethereum's' blockchain. The two words I've avoided explaining in the previous sentence is *Đapp* and *DAO* which I will explain now.

One thing first. In the chapter above I mention The DAO (notice the capital T) which was an venture fund capital organisation calling itself by the name of the concept which I will explore here but it is different. DAO stands for Decentralized Autonomous Organisation and describes any organisation that "[...] *is an organization that is run through rules encoded as computer programs called smart contracts*"[23].

[23] The Decentralized Autonomous Organization and Governance Issues. Regulation of Financial Institutions eJournal: Social Science Research Network (SSRN). 5 December 2017.

Đapp is the term that entails the solidity part of my project. It stands for Decentralized app(lication). The beauty of the name is that the latter Đ looks like a D overlaid with an E but is pronounced Eth in Faroese. The name decentralized app is misguided since it is nothing like the conventional idea of an app but the name was chosen to seem familiar or at least less esoteric to developers.

The idea of a decentralized voting platform traces back to the early days of ethereum development. The use case of a DAO is suggested by Vitalik in his white paper. Let's say that the DAO wants to hold an election. In a democracy everyone gets one vote. But in his example, just like with shareholders in actual companies, you might want to distribute voting power based on how much equity they hold. This is possible with tokens and made easy in Ethereum.

I am more interested in how everyone in society can benefit from a blockchain election system. A group of friends could hold a poll on what day to meet up or what to do. A hairdresser could use it to make people book a time without having to maintain a database. Yes, it could replace the use of a database for public information or having to put up a server to create a website.

In the case of Democracy Manifest (frontend and blockchain combined), KEA could use it as a substitute from paying the usual external survey consultancy with the benefit of openness about the results.

It should be obvious by now that the reason for doing it as a Đapp on an open blockchain is to ensure trust in the code and the final results of the election.

5.1 Structure

One thing I need to point out is that I have more than 1 contract in some files. This enables me to have them interact with each other. In a real life setting I would deploy the contracts incrementally and know exactly what their address is. Because I restart my test network every time I start my computer the given address in the blockchain changes every time. So I deploy related contracts in one sol file so they can call each other. My diagram will signify the ideal object relation between these contracts as separate units.

I have two interfaces which each relate to a blockchain development design pattern for destroying deployed contracts (Killable) and for restricting access to the original deployer (Ownable).

The Authentication contract lets users create accounts and remembers them next time they login.

I have two kinds of elections open and closed depending on if anyone can participate or if it is closed to those whose public key have been added. I've chosen to split them into two separate contracts to respect the design principle called separation of concerns.

As mentioned, I have two ContractFactory contracts within each election file shown as separate entities, since they are separate objects. As the name suggests their job is to create and retrieve contracts.

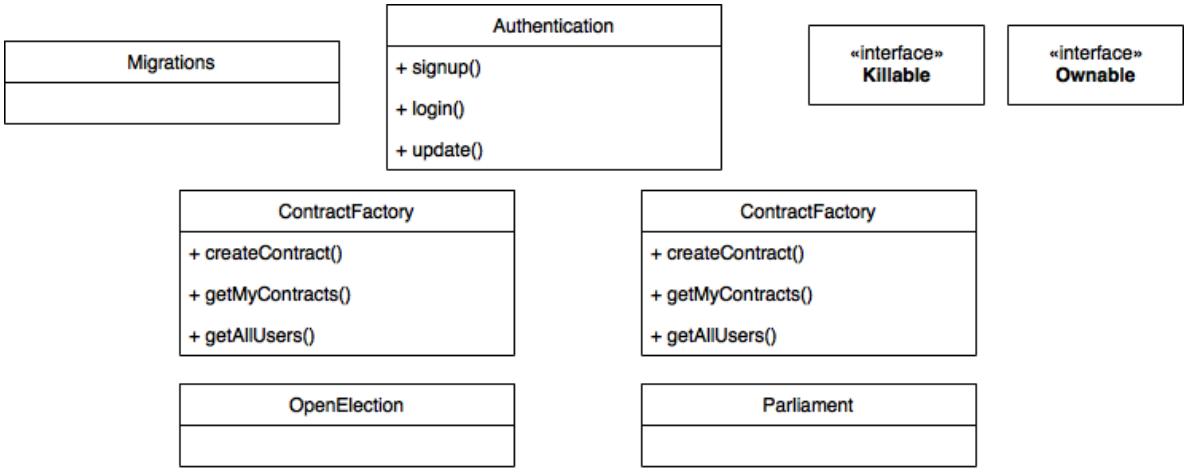


Figure 4: The structure of the solidity part of the Đapp

5.1.1 The two election types

In the website I provide the users with a choice between two election types. OpenElection lets any person with an Ethereum public key vote only once.

Parliament is for closed elections with members added by the contract owner. The contract owner is initially the user deploying the contract. But there exists mechanisms for handing over ownership rights to a new public key.

Since the Parliament election is only for allowed members there are also mechanisms in place for adding members and removing them.

The Solidity struct named Member contains a value for tokens. The member can vote the amount of times on an election as the amount of tokens they receive. This is the principle of the DAO where shareholders might own different portions of a

company and thus have different weight when voting. If nothing else then everyone could get 1 token each.

Another possibility in the Parliament contract is to set a deadline for the vote. This makes sense in many use cases where you want to get the result by a certain timeframe. In the Open contract people are allowed to vote always and if you want the result by a deadline you can simply just check it at that time.

Notation: + (public), - (private), x (owner restricted)

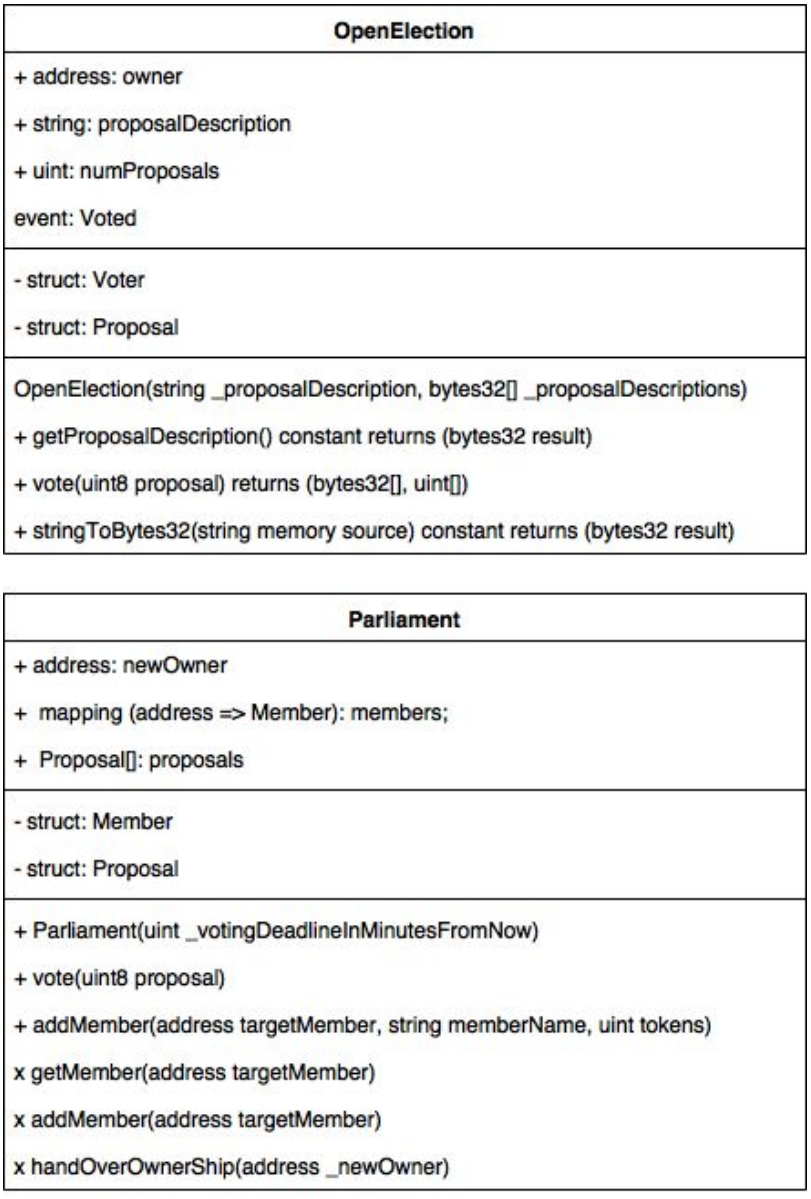


Figure 5: Comparing the two election types and their differing attributes, function definitions. Return type is included cause that is part of the function definition in

Solidity. The first method definition is the constructor which I thought was worth including because of differing parameters.

5.2 Programming quirks of solidity

Solidity looks like a mixture of Java and Javascript though indeed, those are two very different languages. It's both object oriented and shares a lot of syntax with javascript. It's done on purpose to make it easy to get into. There are some things that differ from any non-esoteric language, though, and I would like to get into those. I wish to introduce them by taking parts of my contract code and expounding upon them.

5.2.1 Access modifiers

Some consideration needs to be taken into access modifiers especially when dealing with a blockchain that is accessible to all. The keyword private exists to limit access to only within the contract itself.

Declaring a variable as public creates an accessor for them on the blockchain. Functions are public by default. Code wrapped around the function or in it can further limit access depending on a set of keys or criteria. For more info read the modifier sub-chapter.

5.2.2 Library contracts

Besides contracts there are also library contracts. Here is a definition of a library contract called ArrayUtils:

```
library ArrayUtils {
}
```

Inside a library you can have fully defined static methods. Now inside of your contract you can inherit these methods by adding this line inside the scope:

```
using ArrayUtils for *;
```

The advantage is that instead of duplicating code/functions in each contract you can import a single library and reuse it as much as you like. Abstraction is not just a good practice anymore. It actually becomes costly not to abstract your code when possible.

5.2.3 byte32 vs. string

One of the less nice things about Solidity is that while the datatype string exists a contract can't get the return of other contracts if it is string. Many library contracts return byte32.

Some of the benefits of using byte32 is that it uses less gas because it exactly fits one word of the EVM (32 byte). If the string is longer than then you would have to split it into an array of byte32 strings which is an obvious pain when calling/decoding in frontend.

When giving a string as parameter in a contract where the parameter is defined as byte32 the EVM itself handles conversion. It will convert the string into a byte32 hex string and add zeros as padding. The consolation is that web3 contains methods to convert a string to ASCII and UTF8*: web3.toAscii(hexString);*.

Returning a string is fine if you know that the contract will only be called from outside of the blockchain. It just doesn't seem very smart to rely on this limitation.

5.2.4 mappings

In Solidity there is a data structure called a mapping which is similar to a Hashmap in Java. Below we see the one I have defined in my *ContractFactory* contract.

```
mapping (address => address[]) contracts;
```

This mapping called contracts has an address as a key and an array of addresses as value. The interesting thing about mappings in Solidity is that everything is initialized as 0.

One thing to note is that a mapping can't be looped through. In my contract factory I've handled it by adding the addresses (keys) to a non-public array and making sure that only the factory contract owner (initial deployer) can access it. This way I can look up how many users are registered or retrieve all contracts.

5.2.5 Structs

Structs are data types containing properties defined inside a contract. Unlike an object structs do not contain function. I have several structs in my contracts. One is for the type Proposal that contains what the option is and a count of votes:

```
struct Proposal {
       bytes32 proposition;
       uint voteCount;
}
```

Additionally to structs it is possible to create enums in solidity[24]. In my daily programming I use enums frequently but haven't had the use for it here.

6 DESIGN PATTERNS

Now understanding data types of Solidity we are ready to look at some of the design patterns I employ in Democracy Manifest.

6.1 Restriction of access

In my interface contract the following can be seen:

```
 modifier onlyOwner() {
    if (msg.sender == owner)
      _;
  }
```

A modifier is similar to when implementing an interface on method level as we saw earlier. It's a type of function that wraps a method with code. The syntax "_;" is something amazing that I have never experienced in a language before. It replaces the underscore by injecting the wrapped method inside of the statement.

The above example ensures that the method is never called if the transaction caller isn't the owner. It's also possible to use modifiers to run code after a function has finished returning.

Another way of creating a restriction inside of a function is by doing this (note that the function also takes the above modifier as restriction):

```
function removeMember(address targetMember) onlyOwner {
       require(members[targetMember].exists);
       [...]
}
```

If require returns false it will immediately rollback the transaction and the gas will be returned to caller.

[24] http://solidity.readthedocs.io/en/develop/types.html#enums

6.2 Lifecycles

Everything on the blockchain is forever, right? Not in Ethereum since it is a stateful blockchain. Internal calls exist for destroying deployed contracts. It's through the self-destruct function that takes the *owner* as parameter. Owner is the keyword for the *msg.sender* that deployed the contract in the first place.

Here is my simple killable interface contract. I restrict it on the function level by implementing my Ownable interface and wrapping my function with its onlyOwner function.

```
contract Killable is Ownable {
  function kill() onlyOwner {
       selfdestruct(owner);
  }
}
```

Now I can make any contract self-destructable by simply implementing *Killable*.

6.3 Dynamic deployment

It is required in my voting platform for users through the web interface to create and deploy new Voting contracts. This can be done on two levels.

Had I done it on web level I would need to store the contract addresses in a database. Instead I chose to use the blockchain as my database by having a contract factory deploy then save voting contracts which makes them retrievable.

At first I had my ContractFactory contract in its own separate Sol file but I moved it into the voting contract file. If they share file they also share space on the block and can easily call each other.

If I had wanted to separate them I would have to deploy the Voting contract first and then deploy the ContractFactory with a hard/soft-coded reference to it on the blockchain. This isn't convenient in my development environment where I often start up new blockchains.

Let's have a look at this deployment pattern I've created. I have two attributes. A mapping that takes a user's address and returns an array of voting contracts. By user's address I mean their public address (public key in cryptography) linked to their *wallet* (public/private key-pair).

I've mentioned in the mapping sub-chapter that just like HashMap the data type is not iterable. So I've also added an array of user addresses as a convenience for the site creator to list all open elections. These are retrievable only by the contract owner (original deployer) ensured by my Ownable interface.

For the users there are two methods. *getMyContracts()* returns an array of contracts belonging to the user. It's a bit more complicated in the contract creating function. Here is the function declaration:

```
function createContract(string _proposalDescription, bytes32[]
_proposalDescriptions) returns(OpenElection) {
	[...]
}
```

It takes a string that will be the question that is voted about. It also takes a list of proposals to choose between. And finally we declare that the created contract will be returned.

Inside we create a new contract with the *new* keyword. We push it to the array in the mapping and add the user to the array. An event is fired (which I will go into a bit later) and the created contract is returned.

More could be done to make the election dynamic. Functions to allow users to change the description or new options can be placed in the voting contract. But I feel that this would undermine the integrity of the elections.

7 WEB

7.1 React

To get up and running fast I use the truffle box react-auth which is similar to create-react-app but with truffle configured in. As a React developer its intricacies are second nature to me and in the context of this thesis only how I interact with the blockchain is relevant.

7.2 Redux

Redux is one of the design patterns that can be used with React. React can exist without it. But it boasts of time travel debugging and data safety through immutability.

In Redux there is a so-called store that contains data relevant to several components. If the data is only used inside a component or a inheriting hierarchy of components then it would make sense to leave it in the component state and trickle down through props.

The store has an initial store state. To change the state one would have to go through a flow of first calling an action. The action dispatches to a reducer. The reducer catches the action based on a switch case. Since the store is immutable it is then copied, flattened and returned with the new data element changed.

A great example of how I use Redux as a pattern can be found when a user enters the website. What I want to do then is to initialize the web3 object so I can interact with the blockchain. First I want to check if the user already has a web3 object loaded into the browser by an outside application. If the user doesn't have their own Provider I initialize web3 myself and insert it into the store with my coinbase (wallet info). Remember that in my case the wallet exists in TestRPC and is a combo of public/private key-pair and the Eth coins.

Understanding what I want to do, let's go through the Redux flow. In *index.js* I call the method *getWeb3()* which is wrapped in a Promise and depending on resolve or reject it prints out a success or error log in the console.

The *getWeb3()* method first searches for a web3 object injected into the browser. The two most likely tools to do this are Metamask or Mist. As mentioned earlier Metamask is a wallet that exists as a Chrome browser extension which injects web3 into the browser. Mist is a standalone browser created for viewing Đapps. If the user doesn't have a web3 object injected by a HTTPProvider, then I create a new object. This is how the magic is done:

```
let provider = new Web3.providers.HttpProvider('http://localhost:8545');

web3 = new Web3(provider);
```

For convenience I also initialize my contracts simultaneously as objects so they are callable from any component on the site. I then use the resolve function from the Promise while dispatching my action. My action looks like this where results contains the web3 object:

```
function web3Initialized(results) {
    return {
        type: WEB3_INITIALIZED,
        payload: results
    }
```

```
}
```

Next we have the *web3reducer.js* that contains the switch case which will catch the fired action and switch on the type constant. Finally we flatten the state and add our web3 object:

```
case WEB3_INITIALIZED:
     return {
        ...state,
        web3Instance: action.payload.web3Instance
};
```

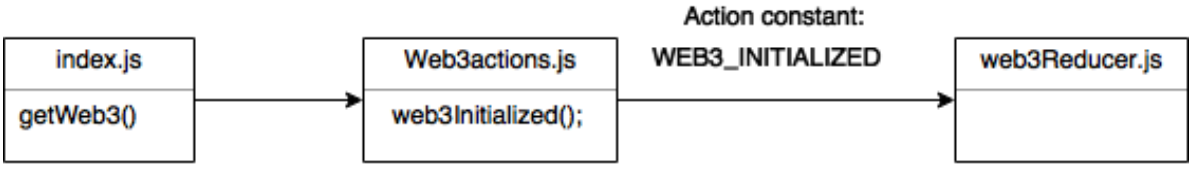


Figure 6: The general flow when getting the web3 object on page load. Not UML.

7.3 Application flow

The signup/login process is great at exemplifying the general application flow and how the React site interacts with the Authentication contract.

Sign-up page: The user is met with a form. They only need to enter the desired username since web3 has retrieved their public key. When submitting the form the method *signUpUser(name)* is called in *SignUpFormActions.js*.

After getting the deployed contract object the action calls the signup method in it:

```
authenticationInstance.signup(name, {from: coinbase})
.then(function (result) {
[...]
```

Note that the coinbase is the public part of a user's wallet (public key + balance) and is always given as parameter in case the transaction requires gas which it does here.

The code snippet above is all that is needed to interact with the blockchain. Remember that the contract is compiled to JSON which is why we can interact with it like this. Web3 ensures that the function is called and data is passed to it.

The signup function in the *Authentication.sol* contract checks if the user already has an account and if not it stores them in an array called users.

The login function is wrapped with a modifier that checks if the user exists and is as simple as that:

```
function login() constant onlyExistingUser returns (bytes32) {
  return (users[msg.sender].name);
}
```

When called by Node, the name is retrieved if the user exists and the user is authorized to enter the admin panel. This entire Authentication system is possible without any database simply by usage of public-key cryptography.

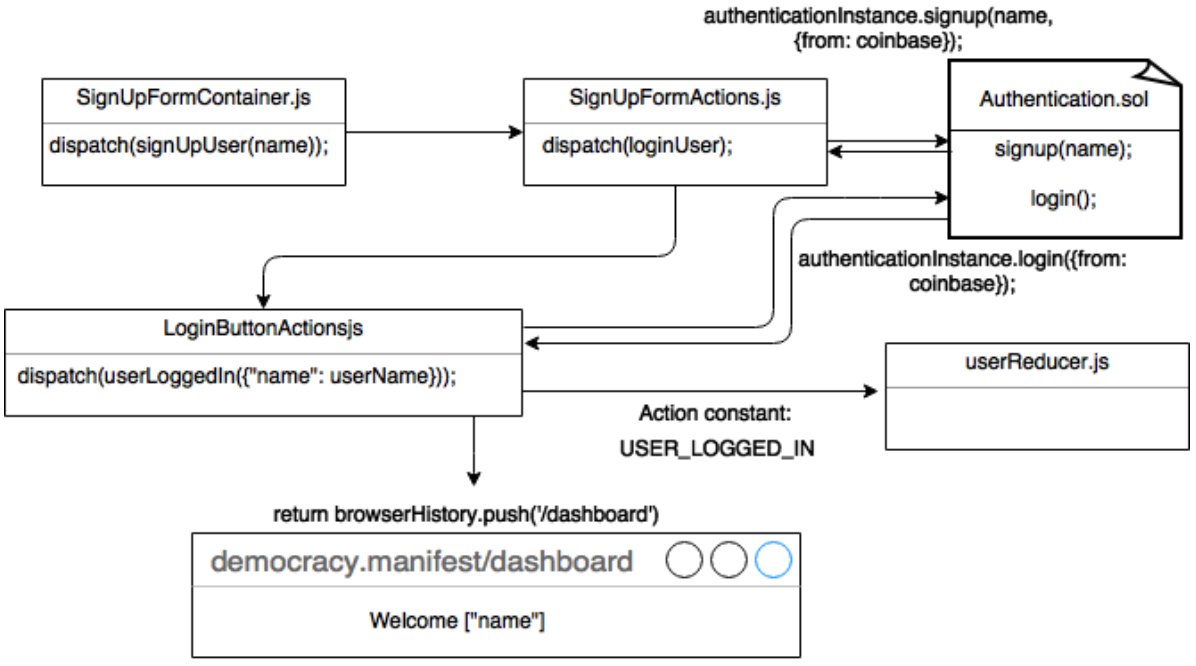


Figure 7: The authentication flow from signing up to automatically logging in and reaching the dashboard. Not UML.

7.3.1 Events

Like above we have many different kind of Democracy actions that lets users create, view and vote for polls. One interesting interaction with the blockchain happens through the concept of events that is part of the EVM. The concept is that you can define events in your contract and then listen to it from external sites. Let's have a look at how I do this.

In my ContractFactory when users create new contracts I might want to listen to these changes and update my frontpage table accordingly. In the contract I have an event defined like the following:

```
event LogContractCreated (address contractAddress);
```

I am following the Solidity coding convention by having the event start with a capital letter and have the first word be Log. Still inside of the contract, inside the create contract function I fire the event and give it the parameter I am interested in:

```
LogContractCreated(electContract);
```

In my Node.js server I define the ContractFactory object and get the event. I also have to define the scope of blocks that I am interested in listening to. Here I define it as all the blocks.

```
let event =
ContractFactory.LogContractCreated({_from:web3.eth.coinbase},{fromBlock: 0,
toBlock: 'latest'});
```

Now I have the event object. I still am not listening to events. To do so I would have to do the following:

```
event.watch(function(error, result){
    if (!error)
    console.log(result);
});
```

Of course, instead of logging the result I could call methods that do something with the parameter.

And finally to stop listening for events the following can be done:

```
event.stopWatching();
```

7.4 Screenshots

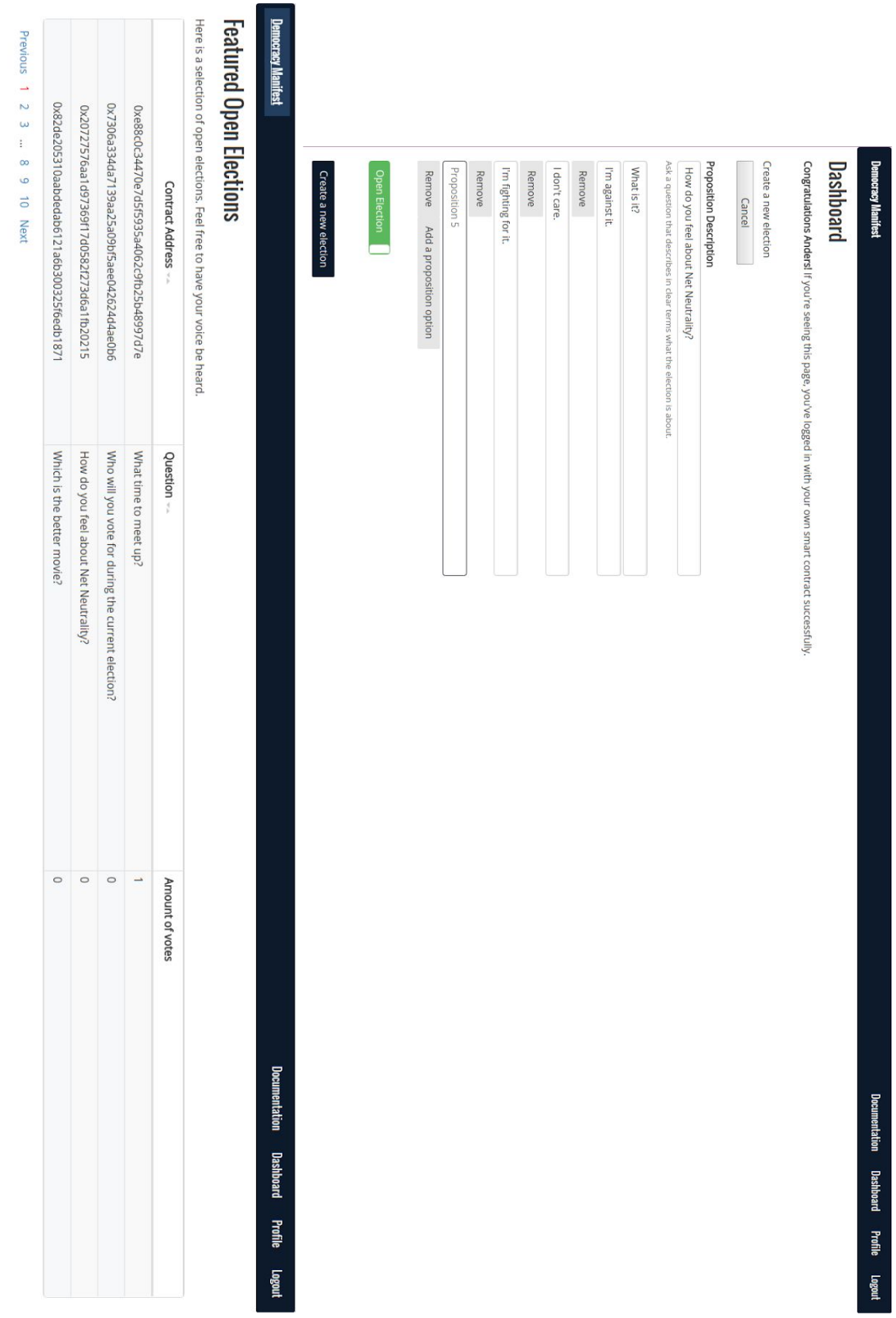


Figure 8, 9: Screenshots of featured open elections and election creation page.

8 TESTS

Two levels of testing can be performed on Democracy Manifest. One is on the contract level and the other is for the Node application.

8.1 Truffle test

There is a specific test contract in my *src/test* folder called *TestAuthentication.sol*. In it I have a reference to my Authentication contract. It uses the Truffle assertion library under "*truffle/Assert.sol*" which is part of the Truffle testing suite[25].

It retrieves the Authentication contract by calling the internally maintained contract - "*truffle/DeployedAddresses.sol*" - that saves the deployed addresses corresponding to the contract.

It then creates a user with the name *'testuser'*:

```
Authentication authentication = Authentication(DeployedAddresses.Authentication());

authentication.signup('testuser');
```

Now I am about to login the user and testing whether they have been created or not. Remember from earlier that a user can only be logged in if they exist because of the modifier around the function:

```
Assert.equal(authentication.login(), expected, "It should sign up and log in a user.");
```

A javascript file makes sure to deploy this test, signup the user, log them in and using assert to test the value. Finally only the following command is required to call the tests:

```
$ truffle test
```

Here is the output:

[25] http://truffleframework.com/docs/getting_started/solidity-tests

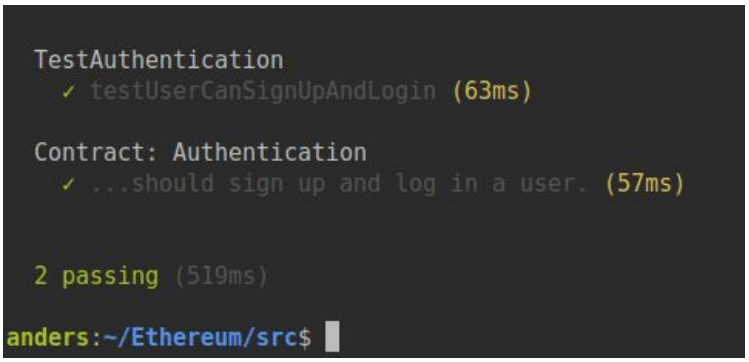


Figure 10: Truffle test output. The authentication contract works. We can signup and login.

8.2 Node Test

For the Node application and React website Jest is set up which is a Javascript testing framework. I have a very simple test in the *src/src/test* folder that makes sure that the website is able to run. With the framework in place more tests can be made as the platform grows.

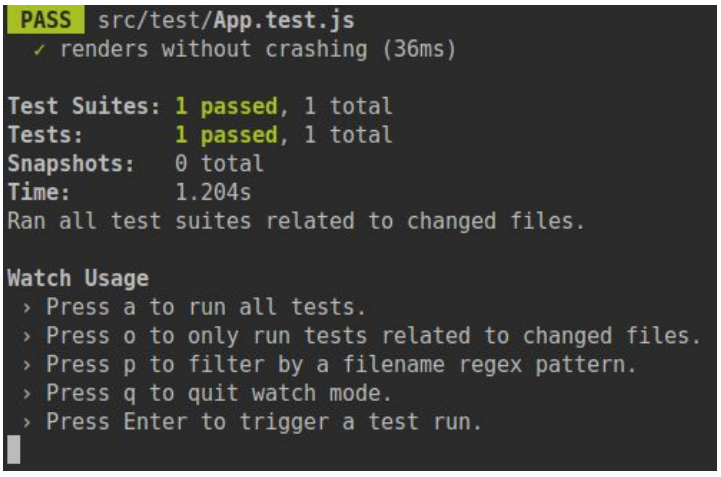


Figure 11: Jest test output. The React page can be rendered as a web page and the store setup is correct.

9 DOCUMENTATION

Ethereum has a cross-language specification for documentation called the *Ethereum Natural Specification Format*[26]. This includes annotation such as: @title, @author, @param, @notice, @dev.

[26] https://github.com/ethereum/wiki/wiki/Ethereum-Natural-Specification-Format

As mentioned earlier, I value documentation and my website features the contract documentations. There exists two node modules for generating documentation. Doxity simply doesn't work and hasn't been updated for the times. Solidity-docs is a low-tech solution which I have been necessitated to use. I have also considered creating my own documentation framework.

I have created a Node script that calls a bash script I have written to generate docs. The shell file looks like this:

```
rm /docs/*

for f in $(find ./contracts -name '*.sol');
       do
       name=$(basename "$f" ".sol")
       echo "Creating md doc file for $name at: $f"
       solidity-doc generate $f > ../src/docs/$name.md

done
```

This puts my md files into a folder called docs which I render in my React app. Now I have full transparency about my contract code besides my public Github repo.

10 CONSIDERATIONS ABOUT ETHEREUM AND BLOCKCHAIN

10.1 Discussions in the community

There are some debates that has the general blockchain community split.

Public vs. private. The initial idea was to create open chains and many in the community believe that privatizing them is against the main idea. But we are seeing several attempts to create private blockchains.

One of the most ambitious projects to create the technology for private chains is called Hyperledger and is handled by The Linux Foundation (who also maintain Node.js). IBM is investing heavily in developing Hyperledgers since the finance world is very keen to implement this technology privately. Private blockchains is also rumored to revolutionize the logistics industry.

Immutability vs. mutability. Many blockchain purists believe that blockchains should only follow their protocol and not be manipulated by human influence. But we are also seeing that EU is passing laws about data and user protection. "*The right to be forgotten*" goes against what blockchains do so the question is how will this affect blockchains? Will laws be passed in the future that necessitate blockchains to be

mutable? How will the community react? Will immutable blockchains be forced to go "underground"?

Ethereum is split on the immutability issue too. I mentioned the DAO hack which has caused the Ethereum blockchain being split and two coins existing: ETC vs ETH.

10.2 Criticisms[27]

I admit that this paper is leaning on the very positive side when talking about the technology. Beside the points expounded upon in the previous chapters I would like to dedicate this space to go through some valid criticism of blockchains and Ethereum.

A lot of hype surrounds all blockchain technology. A lot of people fail to understand the core principles and are thus critical of its actual value for no good reason. Some people see its beauty and extol the tech even going as far proclaiming that everything will/should be on the blockchain. Then there is a very small group of people who understand the technology so well that they see its limits and inherent flaws.

Human limitations exist, of course, such as that the technology relies on users not losing their private keys or all will be lost for them. But that problem exists for any fiat valuta. In the next sub-chapters I will only focus on criticisms that is aimed at the protocol.

10.2.1 Flaws and security threats

Below I scratch the surface on some of the security threats that are specific to blockchains. This leaves out obvious threats such as susceptibility to packet sniffing and DoS attacks.

10.2.1.1 Sybil attacks

Sybil attacks happen on the network level. In short it is when "*hijacking your entire internet connection and connecting you to a fake network.*"[28] There are of course validation techniques to minimize the risk. The easiest way to perform a sybil attack is by creating a honey pot and creating a MITM attack (man in the middle). MITM

[27] I invite the reader to think of ways to improve on Ethereum or blockchain technologies in general and either contact me or contact the Ethereum Foundation directly: https://www.ethereum.org/foundation. Contributed efforts are crucial to human progress.
[28] https://bitcoinj.github.io/security-model#confidence-of-confirmed-transactions

requires an unsecured connection. Mobile wallets connecting to public wifis are at risk.

10.2.1.2 Double-spending attacks + Finney attacks

Double spending attacks is an exploit in race conditions and how the network operates. It happens when two invalid transactions are broadcasting simultaneously [29].

Finney attacks are a specific kind of double-spending attacks. It requires the evil miner to find a block that includes their own transaction and withhold this discovery from other nodes. Before anyone else discovers the block the attacker must spend the Bitcoin with some merchant that accepts unconfirmed transactions.

That's why Bitcoin recommends you to wait 30 - 40 minutes for the network to go through enough election cycles before you can trust that you've received your rightful Bitcoin. While merchants are not recommended from accepting Bitcoins right away some would have no choice because of their business model. BitcoinJs documentation[30] suggest three such business under risk:

1. Entertainment websites that provide immediate downloads after the user has paid. A 40 minute waiting period is a bad customer experience.
2. Currency trading where the market is volatile, time sensitive and irreversible.
3. Shops such as supermarkets where the user walks in, gets the goods, pays and walks out. Holding them in the store for 40 minute would be ridiculous.

Alt coins that promise much short election cycles have solved this the issue with the long waiting times but not double spending attacks in general. It's a valid concern.

10.2.2 Scalability[31]

The biggest challenge with all the currently proposed blockchain technologies is scalability. Ethereum is the only one that promises to solve it. The foundation has made some lofty desired progress milestones such as the ability to support Web 3.0 or being able to verify as many as Visa's peak capacity of 56.000 transactions per second[32].

[29] http://eprint.iacr.org/2012/248.pdf
[30] https://bitcoinj.github.io/security-model#confidence-of-confirmed-transactions
[31] Further reading:
https://en.bitcoin.it/wiki/Scalability
www.coindesk.com/information/will-ethereum-scale/
[32] https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf

The table below gives a better overview of the planned milestones for Ethereum and how they can improve scaling:

Upgrade	Description	On/Off Chain	Scaling Improvement	# Known People Working on It	Status	Optimistic Launch Date
Increase gas limit	Miners increase gas limit	On chain	2-8x	2-10	Waiting for miner adoption	mid 2017
Parallel process transactions	Transactions can be processed by nodes simultaneously instead of one by one	On chain	2-8x	1-3	Concept	late 2017
Swap virtual machines	Swapping the Ethereum virtual machine (EVM) for web assembly (WASM)	On chain	3-10x	1-2	Code in progress	2018
Proof of stake	Changing consensus algorithm from proof of work to proof of stake	On chain	2-5x	3-5	Code in progress	late 2018
Payment channel networks	Parties exchange signed transactions off chain, allowing infinite transactions with only an initial and closing transaction on chain	Off chain	10-100x+	4-8	Code in progress	2018
Truebit	Off-chain smart contract execution through a verification game with dispute resolution	Off chain	10-100x+	2-5	Concept	2018

Plasma	Tree hierarchy of blockchains where only fraud proofs are reported up the tree	On chain	10-100x+	2-5	Concept	2019
Sharding	Validators only need validate some, not all, of the network's transactions	On chain	10-100x+	--	Unlikely before proof of stake	2019

Figure 12: An overview of milestones for Ethereum[33]. The Ethereum Foundation is acutely aware of all criticism and have plans to solve them.

10.3 The future for blockchain development

So much has changed from the first day I wrote a sentence in this document to the moment I handed it in. For instance, the solc compiler has updated from ^0.4.0 to ^0.4.2. Web3 has released a beta that contains a lot of breaking change. Truffle has released a new set of features and implemented Ganache core: a simulated blockchain for testing. And Ethereum itself has released a new version number: 3.0 - Metropolis.

One of the more ambitious ideas surrounding Ethereum is World Wide Web 3. We are current on what is called 2.0. Version 3 is being anticipated as what the original creators envisioned the internet to be. Completely open, decentralized, democratic and not applicable to censorship. It is spearheaded under the name Parity.

Indeed, so much is happening. And Ethereum is a neutral platform that facilitates these changes. It's bound to continue at the same rate in the next few years. New protocols are being developed right now and everyone is free to have their input in the decision making. It's all so thrilling.

We are currently limited in our visions of how blockchains can change the world. Vitalik calls blockchains social institutions in a talk[34]. This opens up a new portal. The proof algorithms implement trust between nodes in a network which is a revolutionary idea.

[33] Provided by the Ethereum Foundation (Creative Commons) with clickable links: https://docs.google.com/spreadsheets/d/1oIaWzybcWLQ22eXTZlLrG7Lhn82ElrRvqi3S5v94l0k/edit#gid=0

[34] Blockchains and Privacy through Strong Cryptography: https://www.youtube.com/watch?v=k-PvB7mwtKI

Surely, it will create a new epoch of trade between humans. Almost like a discovery of a new world. What limits people today is trust. These days, I would never give my money to a Nigerian prince. But with blockchain I wouldn't mind it as long as I have trust in the code. We will chant the slogan "*In code we trust*".

11 CONCLUSION

This thesis is a product of previously undocumented pioneer work. Its simplicity is a virtue of experimentation and testing; constant remaking.

Half of my assignment has been dedicated to explain the moving parts of Ethereum and the other half has been dedicated to the example project called Democracy Manifest. This is intentional since my project is intended to serve as a working example for how to create smart contracts for Ethereum and interact with them. Furthermore, I suggest checking out the appendix that contains my 10 minute guide to create a "Hello world" Đapp.

I believe that I have managed to fully explain what sets Ethereum apart from other blockchains such as bitcoin and the endless possibilities (use case chapter) of creating Đapps.

My scalability chapter should silence all doubt about whether criticism is heeded by the Ethereum Foundation and that solutions are indeed being worked on as we speak. Changes are constantly being made and it's hard to find a footing while everything is so fluid but the Đapp market will flourish in the immediate future.

While I see many problems and nuisances about Đapp development as it is evolving it's a superb exercise for a developer. What I love about developing for the blockchain is that it forces you to program in the smartest way. If you don't then you will be punished. I have explained the concept of Gas and I mentioned The Dao hack. Programmers should flock to languages where good programming and design principles matter.

I feel satisfied that my voting platform Democracy Manifest sates the curiosity of the possibilities and inspires the social impact of Đapp development; of why it matters to have trustworthy and decentralized code. I hope it motivates the reader to try their hands on what should now seem like a less complicated field. Good luck!

12 GLOSSARY

I've tried to keep this paper accessible to people without knowledge about blockchain development but some terms are essential for understanding it. Here are some words I use repeatedly:

ABI (Application Binary Interface)[35]: The interface specification that accepts bytecode and knows what to do with it.
Altcoin: A cryptocurrency running in its own node that is similar to Bitcoin or Litecoin but with some slight modifications.
Blockchain: "is a continuously growing list of records, called blocks, which are linked and secured using cryptography".
Contract: A
DAO (Decentralized Autonomous Organisation): A conceptual
DAO, The : A specific venture capital fund investment organisation that experienced a hack and lost $50 million dollars.
Đapp (Decentralized App): Catchphrase name for an application deployed through contracts.
EVM (Ethereum Virtual Machine): The virtual machine which each client runs that follows the Ethereum protocol specification.
Ether: The coin that is useful for Gas.
Gas: An amount that relates to Ether which is withdrawn when deploying contracts or for transactions on the blockchain.
Gavin Wood: The founding member of Ethereum who has pushed for several innovative ideas in the protocol.
Genesis block: The first block in a blockchain. Alsco called the merkle root.
Merkle Tree: Terminology in the cryptography world for the binary hash tree data structure a blockchain has.
Ethereum Natural Specification Format: Cross-language specification for documentation annotations.
Proof of Excellence: The network rewards those who are in congruence with the most nodes. (See my weather forecast example).
Proof of Brain: Feature of the STEEM blockchain that rewards media content through pre-set variables that define value to the network.
Proof of Stake: A pseudo-random selection of miners are rewarded based on their wealth and age in the network.
Proof of Work: Requiring "work" i.e. CPU power for mining.
Satoshi Nakamoto: The alias of the person who suggested the Bitcoin protocol. Real person unknown.

[35] https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI

Smart Contract: There is a tight and a loose definition of smart contract and I use them interchangeably since the definition itself is changing towards the loose one. In its original sense it is a contract that does something if a certain condition is met. Now it is used for any blockchain contract.
Sol: The solidity filetype.
Solidity: One of the languages to write contracts that can be compiled
TestRPC: Software that creates a local Ethereum blockchain simulation for development.
Vitalik Buterin: The co-founding member of Ethereum who holds the reins today.
Wallet: Originally a cryptocurrency term but can be applied generally for the user's identity. That which contains their public and private key.
Web3: A javascript object that lets you interact with the blockchain through JSON and can pass compiled binary code to the ABI.

13 LITERATURE

13.1 Books

Mastering Bitcoin: Unlocking Digital Cryptocurrencies, Andreas Antonopoulos, O'Reilly Media, 2014

Not many books exist on the topic. None have been written about Ethereum yet.

13.2 Foundational Research Papers[36]

https://bitcoin.org/bitcoin.pdf * - Satoshi Nakamoto - The founding bitcoin and blockchain paper. This paper started the revolution.

https://ethereum.github.io/yellowpaper/paper.pdf * - Dr. Gavin Wood - Yellow Paper

https://github.com/ethereum/wiki/wiki/White-Paper * - Vitalik Buterin - White paper

https://github.com/polkadot-io/polkadotpaper * - Dr. Gavin Wood - Polkadot Paper

https://cdn.hackaday.io/files/10879465447136/Mauve%20Paper%20Vitalik.pdf - Vitalik Buterin - Ethereum 2.0 Mauve Paper

[36] * means that the document will be provided with the handed in thesis.

13.3 Other Research Papers

https://www-935.ibm.com/services/studies/csuite/blockchain/[37]

13.4 Required reading documentation

https://monax.io/docs/tutorials/solidity/

http://solidity.readthedocs.io/en/develop/index.html

http://truffleframework.com/docs/

13.5 Significant Internet Sources

https://github.com/truffle-box/react-auth-box

https://github.com/ethereum/wiki/wiki/Sharding-FAQ

13.6 Tutorials

https://dappsforbeginners.wordpress.com/tutorials/introduction-to-development-on-ethereum/

https://dappsforbeginners.wordpress.com/tutorials/two-party-contracts/

https://ethereum.gitbooks.io/frontier-guide/content/contract_democracy.html

https://github.com/paritytech/parity/wiki/Dapp-Tutorial

https://github.com/paritytech/parity/wiki/Tutorial-Part-I

https://www.ethereum.org/dao

[37] The most thorough quarterly data exposition from IBM with access to the CEO suites (thus the name (C suite study) of the financial world etc. and their shifting relation to blockchain technology. Ibm in investing heavily in hyperledgers which deserve their own paper.

14 APPENDIX I - 10 MINUTE GUIDE TO CREATE YOUR HELLOWORLD DAPP

As this paper shows this field can quickly become complicated. That's why I want to write a 10 minute guide to create your first dApp that everyone should be able to follow.

Install testrpc. This will be your simulated Ethereum blockchain:

```
$ npm install -g ethereumjs-testrpc
```

Install truffle. This is your framework for deployments and migrations.

```
$ npm install -g truffle
```

In your new folder with project name run this command:

```
$  truffle init
```

It will create a new truffle project with contracts, application setup for testrpc on the correct port.

In the */contracts* folder create a new .sol file called HelloWorld and write the following:

```
pragma solidity ^0.4.2;

contract HelloWorld {

	function helloWorld() returns(byte32 message) {
		return "Hello World";
	}
}
```

You should now add your new contract and include it in the *2_deploy_contracts.js* file under the *migrations* folder as:

```
let HelloWorld = artifacts.require("./HelloWorld.sol");

deployer.deploy(HelloWorld );
```

To compile all the contracts:

```
$ truffle compile
```

Now run your simulated blockchain:

```
$ testrpc
```

To deploy the contracts to the blockchain:

```
$ truffle migrate
```

Just like how in database migration frameworks where they save in a new table in the DB the last migration the Truffle framework saves this information in a deployed migration file.

You can interact with your deployed contract in truffle console. Type:

```
$ truffle
```

Truffle console uses web3. To call a method and print its result you could do this:

```
HelloWorld.deployed().then(contract => contract.helloWorld.call().then(console.log));
```

Done! You've created, deployed and seen the result of your first contract.