

Imperial College London  
*Department of Computing*

# **Xcoin, building communities around cryptocurrencies on the Ethereum network**

*Author: Theodor Port*

*Supervisor: Dr. Michael Huth*

*Submitted in partial fulfilment of the requirements for the MSc degree in Msc. Computing of  
Imperial College London*

*September 2017*

## **Abstract**

Large scale western society today is organised around nation states. Most governments are elected democratically by the population but recent financial instabilities have led to a diminished level of trust in governing institutions, leading to the creation of the now popular Bitcoin cryptocurrency that uses decentralized ledger technology and consensus algorithms to achieve trust in a currency without a central backer. In addition, attempts have been made to create full scale decentralized autonomous organisations, or DAOs in short, which in addition to controlling money supply also replace other roles that traditionally require trust or monitoring. These roles include counting votes and fulfilling contractual agreements. A DAO consists entirely of code and is secured by being included in a public decentralized ledger.

Xcoin attempts to provide a platform to connect people with the same ideals and ideas about fiscal and monetary policies. Any policy that is not dependent on geolocation, like a minimum wage policy, could be implemented in virtual economies which will free citizens from having to support their government's policies that they did not vote for. The hope is that Xcoin will allow people to experiment with different ideas and observe the effects they have on an economy, in addition to following passionate ideals they might have like supporting reforestation through the use of their currency.

A full stack web application has been built combining Web2.0 and Web3.0 technology, achieving all basic requirements for organising a community around a cryptocurrency token and allowing it to progress. The main objective was to empower communities to make their own rules, for which a system of smart contracts was created that allows a community to publicise an update they would like to implement in their cryptocurrency, and consequently hire a software developer that expresses interest in creating the necessary code for it. A solid basic structure for the platform was created and there is more room for additional features to simplify the operations of an online community without a central governing body.

## **Acknowledgements**

I would like to thank my supervisor, Michael Huth, for his support in this project and for introducing me to concepts in cryptography and cryptocurrencies during his course in Cryptography Engineering, which was thoroughly enjoyed by many of his students.

# Contents

1: Introduction.....	1
1.1 Introduction.....	1
1.2 Report Outline .....	3
2: Background.....	4
2.1 Money .....	4
2.1.1 What is Money? .....	4
2.1.2 A history of monetary policy and Central Banks .....	4
2.2 Cryptocurrencies and blockchain .....	6
2.2.1 Bitcoin.....	6
2.2.2 Ethereum .....	7
2.2.3 Consensus Algorithms .....	9
2.2.4 DAOs .....	10
2.2.5 ICOs.....	12
2.2.6 Software Developer Job Market.....	13
3: Design .....	15
3.1 Xcoin concept .....	15
3.2 Web development and Ethereum: Web 3.0 .....	17
3.3 Databases .....	19
3.4 Events .....	20
3.4 Smart Contract Design.....	22
3.5 Smart contract best practices .....	23
3.5.1 Send() vs. transfer() vs. call.value() .....	23
3.5.2 Race conditions .....	24
3.5.3 Upgrading a faulty contract.....	24
3.5.4 Timestamp dependence.....	24
3.5.5 General good practices.....	25
3.5.6 Error handling.....	25
3.6.6 Summary .....	26
4.1 Development Process.....	27
4.1.1 Ethereum clients .....	27
4.1.2 Frameworks and Tools .....	28
4.2 Solidity Contracts.....	30
4.2.1 Final Architecture .....	30
4.2.2 Contracts in detail .....	31
4.2.3 Testing .....	35

5: Deployment to the Ethereum network.....	38
5.1 Networks .....	38
5.2 Metamask.....	39
5.3 Examples of Xcoin UI .....	43
6: Evaluation.....	46
6.1 Review of objectives.....	46
6.2 Potential problems .....	47
6.3 Gas costs.....	49
6.4 Future work .....	50
6.4.1 Improved UI.....	50
6.4.2 Optimisation of smart contracts .....	50
6.4.3 Including a modifier <i>byVote()</i> in the Token and TokenManager contracts. ....	50
6.4.4 Allowing developers to update the forum itself .....	50
6.4.5 Checking user input and cleaning the directory structure .....	50
6.4.6 Leaving Ethereum behind.....	51
7: Conclusion .....	52
References:.....	53
Appendix.....	54
A.....	54

# 1: Introduction

## 1.1 Introduction

One of the main inspirations for this project is Andreas Antonopoulos, a bitcoin pioneer, who in one of his many presentations about cryptocurrencies said:

“Ether is not competing with bitcoin; bitcoin is not competing with Litecoin. They are all means to express the transactional modality we want to use at any point in time to achieve our goals. [...] Adoption is not simply the act of using the currency; it’s also attaching oneself to a community that has also chosen to adopt that currency. [...] I am choosing my politics through my currency, and through that choice I am associating myself with a global community that has made the same choice as me, and that is expressing that choice through currency. [...] You want inflation? Use an inflationary currency. You’re a goldbug? Use a deflationary currency. You want a currency that creates a guaranteed minimum income for the poor? Use a currency that expresses those politics. You want a currency that puts aside tokens for carbon sequestration? Use a currency that expresses your green politics. We’re going to start seeing communities, politics, and currencies converge and allow us to make these choices.” [1]

Although not as seemingly simple as he describes it, the potential of a future in which humans reorganize into virtual communities based on currencies is exciting and worth exploring further. Xcoin intends to create an online platform where users can easily create customizable cryptocurrencies that are linked to forums in which users can discuss ideas surrounding the organisation of the community and the features of the currency. Monetary policy has been a tool for the organisation of communities for many centuries, where the first uses were by the sovereign to control the population, and later uses were by the population to control the sovereign. To avoid confusion with fiat currencies, the word used for currencies created in Xcoin will be token.

Today, we live in nations whose economies are fixed to a single fiat currency, and by living in a nation and supporting its economy, we support monetary and other policies that are determined by a political party who is democratically elected every set amount of years. Democracy means that if not part of the majority, a person will be forced to support policies they don’t agree with, and while this is necessary and acceptable for some aspects of government, it is unnecessary for others.

For policies that are based on a persons’ geolocation such as immigration laws, security laws or taxation for public infrastructure, neighbours will have to continue to live under the same rules, as they are both equally affected by them. It would be very difficult, for example, to provide different levels of security to different citizens. However, for those policies that are not based on geolocation, such as redistribution of wealth, the management of money supply, and economic interventions like minimum wage legislations, subsidies and price caps, there is no reason why neighbours should have to play according to the same rules.

As an example, there are two currencies, A and B. Currency A supports banana farmers through education and a minimum wage, while currency B does not. Most sensible farmers will now demand to be paid in currency A, and anyone else who cares about supporting banana farmers will build their business on currency A or demand their salary in currency A and therefore make the virtual economy of currency A grow. If no one cares about farmers, and chooses other benefits associated with currency B, currency A will be valueless and fail, and farmers will be forced to accept currency B.

In some cases, consumers can already exercise their consumer-power today by e.g. buying “fairtrade” bananas, but they still pay for the banana in their respective fiat currency, which in turn supports an economy and government that could neutralise that support by exploiting farmers through legislation. Directly choosing to be part of a virtual economy seems to be more efficient and straightforward. Instead of indicating support for banana farmers through an infrequent democratic election of a party that may or may not fulfil their promises, anyone can start operating in currency A and directly realise their support.

The farmers’ case is of course only an example and many other examples exist. E.g., an acute question today is whether to enforce austerity in times of recession or whether to let more money flow into the economy. Two different currencies could support either side of it, and the members of these currencies will then have to bear the consequences of their decision. This project does not promote any particular economic or political views, but instead aims to create a platform where people with the same views can come together and create the community they envisage. For this reason, the focus of this project is to make the new currencies as flexible as possible. All parameters and methods of the currency can be customized, new ones can be added, and all important decisions are made through a system of voting.

After flexibility, the secondary focus is to make all operations as decentralized as possible. This means that no single entity exists that resides over money supply, storage of data or counting of votes. Instead, all these operations are implemented on a decentralized ledger, which takes away a single point of failure as it exists in many traditional companies and institutions today. In terms of social systems, there are many cases where decentralization is less efficient than the centralized alternative, like consulting experts on a medical matter, and it should not be considered as the unquestionable answer to all problems, but merely work as a tool to achieve freedom from corrupt institutions and to reduce procedural inefficiencies.

The technology that will be used to organise the Xcoin online communities without the need of a central governing body is blockchain, the underlying technology of bitcoin and other cryptocurrencies. Due to its capability of supporting complex smart contracts and its large online community for developers, Ethereum was chosen as the blockchain-based network for this project.

The aims of the project are therefore to:

1. Introduce ideas of monetary policy and central banks
2. Analyse concepts of blockchain-based networks with a focus on Ethereum.
3. Describe the idea of a decentralized autonomous organisation, drawing parallels between Xcoin and theDAO, and how symptomatic issues in DAOs will be avoided
4. Design a prototype of the application with focus on functionality, user friendliness and decentralized structuring
5. Deploy application to a private network that simulates the behaviour of an ethereum network for development.

6. Deploy application to the Ethereum testnet and make site publicly available
7. Review objectives, list potential problems and state what differentiates Xcoin from similar projects

## 1.2 Report Outline

### Background

- A brief history of money and central banks is given to provide context for cryptocurrencies
- theDAO and its issues are described and analysed in detail, focussing on voter apathy as the main issue to also affect Xcoin
- Bitcoin and Ethereum are introduced, providing technical details including consensus algorithms for future implementations of this project
- The legal status of ICOs is analysed and how it may apply to tokens created in Xcoin

### Design

- An overview of the intended functions of Xcoin is given
- The prototype architecture is discussed in relation to functional requirements and security concerns
- Best practices in Ethereum Smart Contracts are described
- Design choices regarding web development are listed in relation to how they fulfil the application's requirements

### Implementation

- Tools and frameworks that were used during development are listed
- Technical details of how the design requirements were implemented are given, mentioning any necessary changes

### Deployment

- The TestRPC private network and the Ropsten test network are discussed and the deployment procedure described

### Evaluation

- General Issues are discussed, mentioning how the project has tried to address them.
- Achievements are evaluated in relation to project aims described in introductory chapter
- Gas usage of application is measured
- Opportunities for future work are listed



## 2: Background

### 2.1 Money

#### 2.1.1 What is Money?

The widely adopted understanding of money in economics is that it has evolved from barter. Before money, when producer of item A wanted item B, they had to find a producer of item B that wanted item A and exchange the goods. While inefficient, this was possible in small communities where transactions were simple. But as communities continued to grow and produce a more diverse set of goods, barter wouldn't suffice anymore. Item B would be ready for harvest at a different time than item A and it was more difficult to find someone that needed exactly what you produced. At that point, the idea of using a medium of exchange emerged. It could be anything: dried cod in Newfoundland, shells in India or tobacco in Virginia, as long as it was portable and universally accepted as a method of payment. The most popular materials however became gold and silver, as they are portable, durable, malleable and rare. This theory about the origin of money has been articulated by Aristotle, John Locke and Adam Smith, author of *Wealth of Nations*, the first modern work of economics. While this understanding of money seems valid at first glance, mainstream anthropology sees the barter story as false and Felix Martin explains why in his book "Money, the unauthorized biography". According to Martin, there is no evidence that barter was ever in fact used as the dominant mode of transaction. Before money, rigid social relationships were in place where social obligations were reliably followed and movement within the social hierarchy was very difficult. This meant that barter or money was unnecessary, as everyone fulfilled their duties as a part of that society and in exchange received the rewards that were associated with it. Goods that were not necessary for survival would be exchanged through war loot, gift exchanges or sacrificial sharing rituals.

Money was not invented to overcome the impracticality of barter, but as a measure of economic value that would be used to record credits and balances. Collectibles, such as coins or shells, would then be used to settle outstanding debts. The point he is making, is that money only works to measure value, rather than being the value itself. Fixing it to any kind of material standard therefore makes no sense, as the economic value that an economy produces is inherently variable, and cannot be fixed to a rigid unit like gold. The question of course is how to set the standard for the currency. Today, it is central banks that dictate the standard according to inflation targets set by the government and they do so by controlling interest rates and the supply of money.

#### 2.1.2 A history of monetary policy and Central Banks

One of the first central banks, in the way that they are known today, is the Bank of England, which was founded in 1694. At that time, many country banks would issue their own banknotes. The reliability of these banknotes depended on whether the public trusted the bank or not, which was determined by their reputation and size. The Bank of England was founded to manage the sovereign's debts, by taking money that investors had subscribed to the bank and lending it to the government. In return, the bank received the right to issue banknotes in the sovereign's name,

possibly the most reliable of all. Before this arrangement, the sovereign was facing real competition from private banks, which were much better at creating an international system of borrowing and lending, and whose banknotes had such a good reputation that they threatened the power of the sovereign. The merging of government and bank therefore benefitted both sides and remains the bedrock of the modern monetary world.

However, the question about the standard of this new public-private money was still subject of debate. When by the early 1690's, almost all silver coins had disappeared from the market as people were melting the silver that has become more valuable than what the coin represented, William Lowndes suggested to reduce the amount of silver contained in a coin. A seemingly sensible solution, it wasn't implemented after John Locke was invited to offer his opinion on this matter and, given his prestige, convinced the government otherwise. According to Locke, a 'pound' was nothing but a reference to a definite weight in silver and therefore could not retain its value after losing some of its silver content. Locke was coming from the point of view of political Liberalism, which tried to make citizens more independent from the sovereign. According to him, making the standard of a silver coin flexible meant that the sovereign could declare the coin as valueless at any time, while fixing the standard to the weight of silver gave citizens the guarantee that the coin would retain its value. If government decided to devalue the coin, owners could just sell the silver for the same price. Although well intended, it was problematic as John Law would argue 20 years later.

John Law was a brilliant Scottish gambler turned economist, who believed that the monetary standard should be adjustable to ensure a sufficient supply of money. Money, he argued, would not create a healthy economy by itself and the more people used money as a security, the less money was in circulation to create wealth and to facilitate mobility. Confusing the representative token with the technology itself and hence giving it a fixed standard would encourage the hoarding of money as a security and ultimately harm the economy and therefore the owners of the token themselves.

As a solution, he suggested that the sovereign issuer should have the ability to adapt the supply of money to the needs of private commerce and public finance. As supply and demand dictate a currency's value, this would also adjust its standard.

John Law didn't manage to have a sustained impact on monetary policy at his time, but his views resonate with many economists to this day and it is generally accepted that the monetary standard should indeed be adjustable by controlling the supply of money. The silver standard in England moved to a gold standard until it was finally abolished after the First World War, and today central banks are in charge of setting the monetary standard.

While monetary policy is economic policy enacted through central banks, fiscal policy is economic policy enacted through government and simplified examples of how both can directly affect the underlying community are:

1. Lower interest rates on loans and mortgages encourage business owners to expand their businesses as more funds are available to borrow. Reduced mortgage fees will give home owners more money to spend
2. Minimum wage reduces income inequality, but can also lead to job losses
3. Change in taxes affects the incentive to work
4. Subsidies help a specific industry develop faster at the cost of other competing industries

In summary, most economists believe that money is a representative token of economic value, rather than being that value itself. Its standard, rather than being fixed, should be flexible and

reflect the needs and requirements of the people and institutions it serves, although opinions differ on how much governments should be able to interfere into the economy. Today, in the U.K., the standard is set by a democratically elected government and maintained by central banks through controlling the supply of money and changing the interest rates for which loans from the central bank are made. Monetary and fiscal policies are both powerful tools and can benefit their communities greatly when managed well.

## **2.2 Cryptocurrencies and blockchain**

### **2.2.1 Bitcoin**

Bitcoin was the first successful digital currency and today has a market value of \$44bn with over \$1bn in daily transactions. It was created shortly after the financial crash in 2008, when distrust in traditional fiat currencies was at a peak, making the world more open for alternatives. The currency was created by the unknown Satoshi Nakamoto, who used cryptographic concepts to create a system in which users would work together to maintain a distributed ledger, which keeps a record of validated transactions. Although this was no new concept, Nakamoto managed to solve the problem of double spending, which previous cryptocurrencies didn't manage to do.

He did this by letting nodes solve cryptographic puzzles in order to gain the privilege of adding the next block to the distributed ledger (blockchain) and earning a reward in bitcoins. The cryptographic puzzle consumes CPU power to solve and is also known as proof of work, as anyone can calculate how much computing power (work) would be needed to change the last X amount of blocks. Although each node needs to store the entire blockchain in memory, having a distributed ledger gives the massive advantage of taking away the power and vulnerability of one centralised storage for data. In other words, the power of granting access to and modifying the data does not lie in the hands of a single entity, and there doesn't exist a single point of attack, which makes it much more difficult for an attacker to retrieve large amounts of information in a single attack.

However, problems still exist and currently only five mining pools, which are groups of nodes that combine their CPU power to increase their chances of mining a block, possess more than 51% of the total CPU power and could therefore work together to perform a 51% attack, in which they can redirect previously spent transactions back to their own wallets. Furthermore, the bitcoin community is divided on whether to implement certain updates, primarily to do with scalability, which could lead to a hard fork, in which basically to versions of the blockchain co-exist, both recording the same transactions. This lowers trust, which is arguably the core value of a currency.

Bitcoin is further limited by the simplicity of the stack-based programming language it uses. Initially, the idea was to prevent complicated bugs and attacks, as a simple language can only do so many things, but it is now becoming clear that other blockchain systems, like Ethereum, have an advantage by being able to construct much more complicated series of transaction, known as smart contracts, which trigger actions based on logical conditions.

In bitcoin, the blockchain records the transactions that arrived at the successful mining node in a time-window of ten minutes. Each node itself then keeps an updated list of Unspent Transaction

Outputs (UTXOs), and whenever a spender references a previous transaction output (txout) as an input to a new transaction, each node verifies that this transaction is indeed part of the UTXOs, and that the user possesses the private key that belongs to the public key, to which the transaction was locked. What is known as an address in bitcoin is just the hash of a user's public key.

## 2.2.2 Ethereum

As mentioned previously, Ethereum expands on the concept of bitcoin by allowing for smart contracts, which are written in various Turing-complete languages, the most popular being Solidity. In addition to recording transactions, the blockchain also records account balances, which at first seems to pose an even larger task on each node to store the full state of the blockchain, but as the majority of balances stay constant between blocks, pointers can be used for efficient storage. A special data structure, called a "Patricia tree" is used to implement small changes in the state efficiently. Nodes engage with the Ethereum system through the Ethereum Virtual Machine (EVM), which can be thought of as a virtual computer. Following that logic, it is the state of this virtual computer that is stored on the blockchain, instead of a simple storage of transactions. In other words, the EVM can be thought of as a large computer that executes in every node of the network and whose state is agreed upon by the consensus protocol similar to bitcoin. Again, this might seem like an inefficient version of a cloud computer, but the advantages lie in its decentralized and open operation as was explained for the bitcoin protocol.

The Proof of Work (PoW) algorithm used for ethereum, called Ethash, differs from bitcoin's (SHA256) in that it uses much more memory through a Directed Acyclic Graph, which is a 1GB data structure that needs to be used by every CPU trying to solve the cryptographic puzzle. This design choice was made to prevent massive farms of Application Specific Integrated Circuits (ASICs) to gain large chunks of computing power and therefore make the network less decentralized. Indeed, while mining with a standard private CPU on the bitcoin network is completely pointless, one stands a chance to find a block on the Ethereum network. In addition, while the difficulty of the cryptographic puzzle for bitcoin is constantly adjusted in order to release blocks every ten minutes, the target block release frequency for Ethereum is 12 seconds. This allows for transactions to be confirmed much faster, although with less Proof of Work. Furthermore, the reward for mining a block in bitcoin is halved every 2016 blocks, with a total upper cap at 21 million coins, while it is kept constant at 5 ether per block for Ethereum with an upper cap of 18 million per year.

As mentioned previously, the state system for Ethereum is made up of accounts, where each account has a 20-byte address and where state transitions are the transfer of value in the form of tokens or ether, and information between different accounts. Each account has four fields:

1. A nonce, which is a counter that is incremented after each transaction to make sure each transaction can only be processed once.
2. An account's current ether balance
3. An account's contract code if present
4. An account's storage (empty by default)

Ether is the system's own currency, which acts as a kind of fuel to pay for transactions in terms of how much memory and computing power they take. This is necessary to avoid Denial of Service

attacks or bugs, which could happen through infinite loops. What bitcoin solved by using a simple programming language, Ethereum solved through using ether. Even when implementing new currencies or tokens on the Ethereum network, one will always need ether as a way to pay for the processing of transactions. However, contrary to the Bitcoin network, a neat system could be implemented, in which a contract keeps an ether balance, with which it refunds the amount of ether the sender had to pay for the transaction. It then collects the fees in the new currency from the sender and sells it in an auction for ether to restock its own ether balance. This means that every sender needs an initial amount of ether, which can then be reused for all future transactions and the sender only pays fees in the new currency.

Two types of accounts exist on the EVM: externally owned accounts (EOA) and contract accounts. An EOA is strictly limited to signing and sending transactions to other accounts and are controlled by a private key, much like a wallet in bitcoin. A contract account is controlled by the code within it and responds to inputs from other accounts. It is important to note that a contract must be able to create deterministic results, which means that it cannot create native processes such as generating random numbers. Contrary to the intuitive meaning of “contract”, a contract account acts more like an autonomous agent that responds to its environment by executing its own code. These are the smart contracts. The inputs a contract account receives can either come from an EOA or from another contract account, and are called transactions and messages respectively. Transactions contain:

1. The recipient
2. A signature identifying the sender
3. The amount of ether to be transferred
4. An optional data field
5. A STARTGAS value, representing the max. number of computational steps the transaction can trigger
6. A GASPRICE value, which is the fee in ether the sender pays per unit gas

The data field can be accessed by the EVM with an opcode, which could be used for example to register the name and address of the sender. Usually, each computational step costs 1 gas, but more complicated steps can be more expensive. In addition every byte of the transaction data costs an additional 5 gas.

Messages are sent between contracts and contain the following fields:

1. The sender of the message
2. The recipient of the message
3. The amount of ether to be transferred alongside the message
4. An optional data field
5. A STARTGAS value

A contract account can produce a message by executing the call opcode and the amount of gas supplied by the initial transaction between the EOA and the contract account applies to the total gas consumed by the initial process and all sub-processes.

When a transaction or message runs out of gas, it reverts back to its initial state and the gas is lost. However, parent execution does not revert, which means that contract A can safely call contract B and lose at most the amount of gas it specifies as STARTGAS in the message.

The code in the contract accounts is written in a low-level, stack-based bytecode language, called EVM code. The EVM code has access to the stack, the infinitely expandable memory and long-term storage, where long-term storage is the only one that persists after computation ends.

### 2.2.3 Consensus Algorithms

The fact that Xcoin resides on the Ethereum blockchain means that Xcoin users are dependent on Ethereum miners, which will become problematic if tokens were ever to compete against ether. As one of the main drawbacks, this problem deserves attention and future implementations of Xcoin will give users the opportunity to create their own blockchains. A quick overview of consensus algorithms that are required for mining is therefore appropriate:

#### Hashcash-SHA256<sup>2</sup>

This is the algorithm used by bitcoin. In order to successfully mine a block, a node needs to repeatedly compute the double SHA256 hash of the current block header, changing the nonce, an otherwise useless data field in the block header, between each iteration. If the result is below a certain difficulty level, the node succeeds and receives the privilege to add the next block to the blockchain and collect a reward in bitcoin. The advantage of this algorithm is that it can be made extremely difficult to solve, but extremely easy to verify, which is important to allow non-mining nodes to verify transactions. However, disadvantages include large power consumption and centralization of hashing power in large ASIC farms.

#### Hashcash-Script

This algorithm is used by Litecoin and imposes a 128 kB memory footprint on the CPU trying to solve it, which makes the network less vulnerable to centralization of mining power arising from limited access to ASIC equipment. However, there are counter arguments, as the HashcashSH256<sup>2</sup> algorithm is much simpler, which means that any user can invest their savings into making ASICs. Script ASICs would be much more difficult to build and in the long-term could mean that a company that has access to Script ASICs will completely dominate hashing power. Furthermore, memory-time trade-offs are possible, so it is actually possible to build faster chips and reduce the amount of memory needed to be competitive. Hashcash-Script has a further disadvantage in that it takes significantly longer than Hashcash-SHA256<sup>2</sup> to verify, although the vast amount of CPU work of validation is spent on the verification of ECDSA signatures for every transaction, so the last point might not actually make a big difference.

#### Ethash

This is the consensus algorithm used by Ethereum. Compared to other PoWs, it requires nodes to store a 1GB dataset, and to randomly select slices from it to compute the Keccak-256 hash. This hash needs to be below a certain difficulty, as is the case for bitcoin. To verify, light nodes only need to access a 16MB cache that they can compute by scanning through all block headers up to the current one. The cache and dataset is only updated every 30000 blocks, so miners spend most time searching through the dataset, instead of modifying it.



If enough time exists, further consensus algorithms like Proof of Stake, Proof of Burn or Proof of Concept can be added to the available options when customizing the token

## 2.2.4 DAOs

### theDao

theDAO, short for Decentralized Autonomous Organisation, was a corporation that was started as a system of smart contracts on the Ethereum blockchain that would fund startups based on votes from its more than 18.000 stakeholders. It raised around \$150M million in an ICO and was the most successful crowd funding project up until that point. Albeit this success, many were sceptical as picking the right startup to invest in forms a difficult task, and even well-paid, connected and educated venture capitalists consistently underperform public stock markets. How would a large group of stakeholders, many of whom are not experienced investors, make better decisions, especially when they receive less information about the projects than traditional VCs would have available? These were of course legitimate concerns, but were never proven to be justified or not, as the DAO was victim of a hacker that managed to withdraw over \$50M worth of ether only six weeks after its launch. This hack prompted the Ethereum community to ultimately perform a hard fork, which would split the Ethereum blockchain and revert the new chain to a state before the attack. Although backed by 89% of the community, this was a controversial decision as it went against the principle of ‘code is law’, and some insisted on staying on the original chain, creating Ethereum classic. The hack was also the end of popular interest in the DAO.

The code for the underlying smart contracts was written by the company slock.it, and was designed as the underlying principle for any DAO, not just “theDAO” that failed. As the creators describe in the White Paper, it is designed as the foundation “for automated organizational governance and decision making”. Members purchase the token of the given DAO for ether and in return receive voting rights proportional to the amount of token they possess. Those voting rights can then be used to vote on incoming proposals that are filtered by a democratically elected curator. If a proposal passes a vote, ether is sent to the proposer to implement their idea. This concept can be applied to anything where a contractual agreement is needed. Whether it is a startup in need of funding, a charitable donation, paying a gardener to maintain public parks or updating the very token itself. However, the wide scope brought a lot of confusion with it and the connection between real world applications and smart contracts was still very unclear. Who would check on whether the gardener did his job? Of course, a second party could be hired to do this, but it continues: who checks the checker of the gardener? Or how can investors be guaranteed to be paid their rewards, when the startup provides services and products whose state cannot easily be monitored in a smart contract?

In other words, human trust is still required for many contractual agreements, and the benefits of decentralization start to fade. The DAO had the potential to become anything, while at the same time being nothing, at least not in the context of the current infrastructure of the world, where having an address in hexadecimal format is still a rarity and smart contracts don’t have a clear legal status yet. The creators of The Dao have realised the weaknesses of the project and are now moving on to apply the concept to more narrowly focused applications like “The charity

DAO”, in which holders of DAO tokens can vote on charities that they want to support in a completely transparent way.

Xcoin aims to differ from the DAO in a few ways:

1. The narrative is currency rather than contracts, limiting its scope and giving a clearer image of what users can utilize it for.
2. Xcoin has the powers of a central bank and users are able to implement any money supply policies they like.
3. It is the holders of tokens that come up with updates, instead of the implementer as in the DAO. Updates can be discussed in forums and only when enough interest is indicated through up-voting a topic, should the update be proposed for a vote. This eliminates the need for a curator and aligns updates with the political and economic interests of the community.
4. Communities are built around political and economic ideals, rather than corporate interest. This could have an effect on the problem of voter apathy, as is described next.

### Voter Apathy

During theDAO’s short lifetime another problem crystallised, which was not whether stakeholders could make the right decisions or not, but whether they can make a decision at all. In the weeks of its operation voter participation never exceeded 10%, even when the vote was on an important moratorium to prevent exploitations of identified bugs in the code. This is surprising as one would have imagined higher participation when the people’s own investments are at risk. One of the main reasons for this voter apathy was probably the fact that once voted, users would not be allowed to perform a DAO split anymore. The DAO split was included as a feature to protect minority owners against “Majority robs the minority” attacks. In such an attack, a user holding 51% or more of the voting power could pass a proposal to transfer all remaining tokens to their own accounts. To prevent this, users would be able perform a split, in which they create a second DAO with themselves as curator and transfer all their funds to it, escaping the malicious proposal, but keeping the rights for the rewards of previous successful proposals. A problem with this solution is that it provides an incentive not to participate in a vote, which is exactly what happened. Furthermore, votes require time and effort, which many DAO-token holders weren’t willing to provide. The fact that users could exchange their DAO-token back for the originally paid ether until they participated in a vote meant that theDAO was a very low risk investment opportunity with the potential of huge returns if successful. Many users were probably hoping to get on board the next decentralized success story, without being willing to commit. However, voter apathy cannot only be blamed on those factors, as it continuously proves to be a problem in many other decentralized applications, and as Dan Larimer, CEO of cryptonomex, summarises:

“Fancy technology can obscure our assessment of what is really going on. *The DAO* solves a single problem: the corrupt trustee or administrator. It replaces voluntary compliance with a corporation’s charter under threat of lawsuit, with automated compliance with software defined rules. [...] What *The DAO* doesn’t solve is all of the other problems inherent with any joint venture. These are people problems, economic problems, and political problems.[...]Ultimately, technology can only aid



in communication, it cannot fix the fundamental incompatibilities between individual self-interest and community decision making.” [2]

The incredible financial gains of Bitcoin and the subsequent financial interest in the blockchain space has massively helped to fund its development and expose it to the public. Advertisements for cryptocurrency trading can now even be found on the London tube. Financial incentives are also clearly powerful in reaching consensus in large groups, as can be seen in the fact that today, betting agencies are often the most accurate predictors of political and other events. Robin Hanson even takes it so far as to suggest a form of government called Futarchy, where democracy decides over what we want but betting markets say how to get it [3].

However, the failures of projects like theDAO and BitShares to achieve active member participation also shows that financial interests is not enough to allow a company or organisation to function without a centralized governing body. The reason bitcoin managed to break through was not because people were hoping to make profits from it, at least not in the beginning, but because people were committed to it from an ideological perspective. An activist movement called ‘cypherphunks’ was heavily involved in developing precursors for bitcoin and actively helped bitcoin progress. Hal Finney and Wei Dai, two of its members, have denominations of the currency named after them. The groups ‘manifesto’ makes clear that one of the groups’ primary concern is the need for privacy in an open society. [4]

Another example of ‘success due to ideology’ is Dogecoin, a currency that was partly developed as a joke and derives its name from an internet meme. In January 2017, it reached a peak market cap of \$400M.[5] Voter apathy will almost certainly also pose a challenge to Xcoin, but the hope is that commitment due to ideology will cause people to participate more actively. Only when a base of active members for a particular token is formed will financial interests be helpful for its growth. Furthermore, as members have the choice between different tokens, they are not forced into a single community as was the case with theDAO or BitShares.

Finally, different systems of voting exist and have been proposed to be more efficient and representative of a community’s needs than majority voting. Two examples are quadratic voting, in which the cost per vote increases quadratically [6], and range voting [7], where users give each voting option a score and the option with the highest average wins. Ideally, Xcoin will offer different voting systems, but for the sake of simplicity, majority voting will be the only option available for now.

## 2.2.5 ICOs

ICOs, short of initial coin offerings, have managed to raise over a billion dollars for early stage start-ups in the past year [8]. An ICO is a means of fundraising in which a company attracts investors through releasing their own cryptocurrency token in exchange for fiat currencies or, as is usually the case, in exchange for other cryptocurrencies like bitcoin or ether. Ethereum also started through an ICO in 2014 and raised \$18 million worth of bitcoin [8]. They are a powerful tool for startups to raise money quickly, as they bypass many of the regulations that are typically required by banks and venture capitalists in a fundraising process. This poses a danger for investors and in July 2017, the U.S. Securities and Exchange Commission published a report stating:

“Depending on the facts and circumstances of each individual ICO, the virtual coins or tokens that are offered or sold may be securities. If they are securities, the offer and sale of these virtual coins or tokens in an ICO are subjected to the federal securities laws.” [9]

And in another report:

“Based on the investigation, and under the facts presented, the Commission has determined that DAO Tokens are securities under the Securities Act of 1993 and the Securities Exchange Act of 1934.” [10]

A month later the Canadian Securities Administrators issued a similar report [11].

Security laws required the issuer of the security to provide adequate information to the investor to allow him to make an informed decision. These regulations are important to protect investors from Ponzi schemes and fraud and will slow down the hype around ICOs and hopefully make the landing slightly softer, if the bubble decides to burst. However, if currencies like bitcoin were ever to be considered securities, their mining and exchanging could become very difficult, as issuers like miners would be obliged to provide information that they don't have available. Although no court or government agencies have expressed an opinion on whether bitcoin is a security yet, a paper looking at different securities and comparing them to bitcoin concludes that “bitcoin does not fall within the definition of any common type of security “ and that “in addition, Bitcoin does not appear to fall within the broad definition of ‘investment contract’”[12]. Legislations will continue to adapt to the growing cryptocurrency market, and for Xcoin, members will have to stay informed about the legal status of issuing and selling their token.

## **2.2.6 Software Developer Job Market**

Bontysource is an online funding platform for open-source software, where clients put bounties on issues in open-source software that they would like to see addresses. With an average posting of approximately \$15 thousand worth of bounties [13], it shows that a market exists for modular developer jobs. Although the number of bounties posted is on average much higher than the number of bounties earned, this trend does not show when looking at bounties by price. This means that there is a large amount of small value bounties that are never claimed, whereas most of the high-value bounties end in completed jobs.

Updates in Xcoin tokens will probably constitute much smaller jobs than most of those posted on bontysource, as even the contract for the DAO didn't exceed a thousand lines of code. In combination with a shortage of blockchain developers [14], this will create a bottleneck in the progress of tokens and their communities, and could lead to members quickly losing interest when no developers can be found to implement their ideas. This is an issue to which no solution exists at the moment. The small scale of possible updates means that developers that sympathise with the ideals of the community might be willing to work on it, but the shortage of blockchain developers also means that the first contracts will contain large amounts of bugs that won't be detected by many, which could be dangerous if users start storing valuable tokens in these contracts.

Generally, Xcoin would have to wait some time before the right environment exists to reliably implement updates that are requested by communities. In the meantime, they will have to rely on

developers from within the community or the creator of the project to implement updates for them.

## 3: Design

### 3.1 Xcoin concept

The underlying idea of Xcoin is to create a platform where users come together based on ideas and beliefs they share. When creating a token, users can specify its issuance rate, upper cap, required consensus for successful votes, name and whether it should refund ether spent in transactions in exchange for token. In order to not limit users to these choices, communities can propose ideas for new methods and parameters and hire developers to implement them. The developer will be paid in the given currency, which creates an incentive for him/her to create well designed code, as its quality will directly impact the value of the developer's salary.

One of the defining features of the Ethereum network is that once a contract is created, it cannot be changed as it forms part of the ledger. This means that new contracts that inherit the state of the previous contracts are needed in order to advance a token. In order to agree on a certain update, the community votes, thereby freezing their token savings until the end of the vote.

Users organise themselves in reddit-like forums and anyone can create a new forum for a new token. A user with an idea for an update might find it difficult to put a price tag on it, if they have no background in computer science. For this reason, a public auction, called *BountyHunt* is started in which the user specifies the idea, and developers offer to implement it for a certain amount of token. Ideas that other users find interesting can be up-voted and discussed. To simplify matters, each community can set a fixed hourly wage for developers. This means that developers compete in time, not in price, and it solves a problem discussed in the evaluation. Once a developer published their offer, the community can vote on whether to accept this offer or not. If the threshold to implement an update is reached, a smart contract will be triggered that pays the developer the specified amount, if he/she delivers the code within time and without bugs. The developer will be paid from the balance of the *TokenManager*, and in the case of insufficient funds, the extra funds will be created by the manager, which causes the currency to lose a little value and therefore everyone who possesses the coin at that moment pays for it proportionally to their savings. The size of the salary should be so low in comparison to the total supply that this will not have an effect on the market. Although the payment in local tokens encourages the developer to write clean code, it doesn't solve the problem of a malicious developer incorporating an attack into the update. For this reason, the code must be reviewed before its release, which will be implemented by a process called *BugHunt*, in which the developer uploads the source code, and anyone is rewarded for finding a bug. To prevent developers and bug-hunters to cooperate, the reward for a bug is subtracted from the developer's salary. Once a bug is found, a further vote is needed to certify it. This solution is not optimal as many members will not have the required knowledge to make this decision, but they can be aided by more experienced users in the forums, and it forms a better alternative to having specified 'bug-detectors' that could cooperate with a malicious developer. Ideally, an additional 'in-house' detective exists to review code, in case bug-hunters don't want to invest time in reviewing code when there's the change of no bugs and no reward. Once an update is accepted, its state is initialized to that of the previous contract and the previous contract destroyed.

Between acceptance and destruction, users have a set amount of time to transfer their savings, as this would be too costly for the *TokenManager* on its own. To summarize, there are two groups in the process of an update implementation:

### The community

The complete group of users that own the local currency. Anyone can come up with an idea for an update, but only if a developer and X% of the community come to an agreement will the update be implemented. X represents the percentage of required consensus based on currency holdings, which can be specified individually for each community. The community needs to be protected from malicious developers through the public *BugHunt*. If a bug is found in *BugHunt*, community votes on legitimacy of bug.

### The developer

An independent software engineer, who doesn't need to be part of the community. Needs to be protected from failure of payment by community. Can check the source code of the token to verify the conditions for his/her payment. Bughunters then look at code and can either cause the developer to be paid or get another attempt at fixing the bug. If the developer has exhausted their maximum amount of tries, the update is canceled. The community has very little incentive to cheat a developer as this would massively impact their reputation and the following devaluation of their currency will outweigh the gain of free code. Developer gets to know about updates through public auction.

Figure 1 shows a simplified version of the Xcoin platform and its operations. For detailed lifecycle of an update implementation please refer to Appendix A

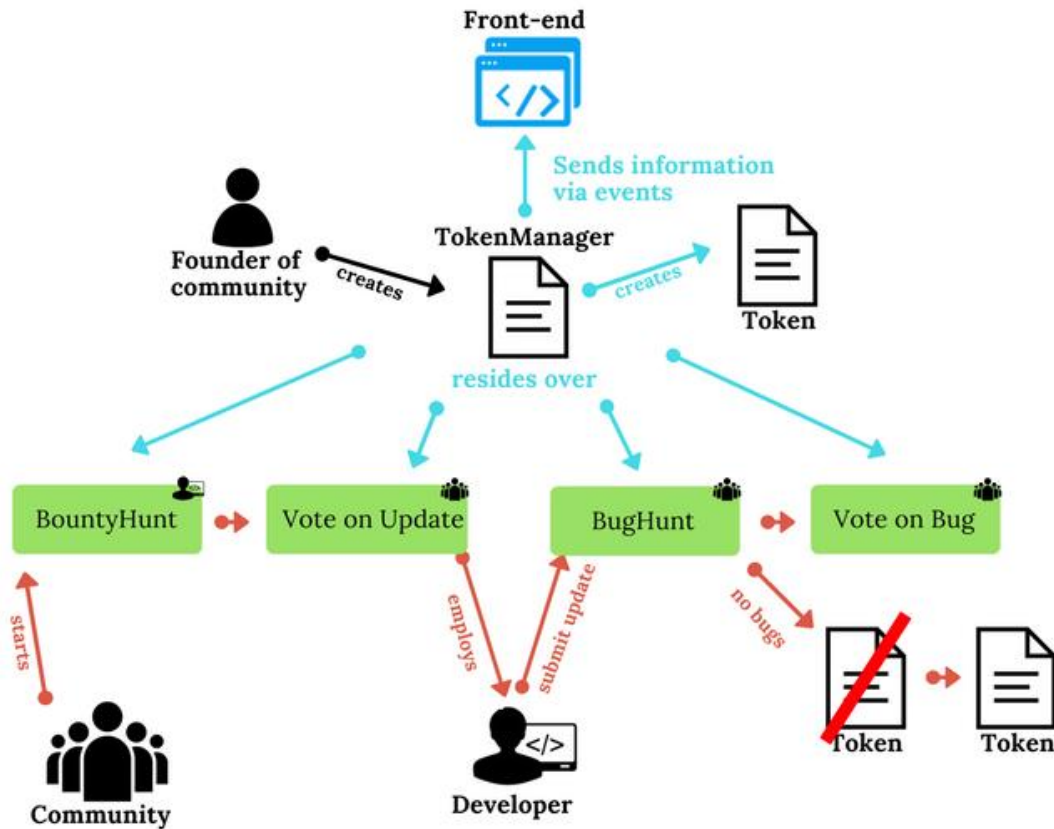


Figure 1, Overview of Xcoin functionalities and relationship between stakeholders

In order to relieve users from having to manage an ether wallet for blockchain transactions, the *TokenManager* includes an additional feature that allows users to pay for blockchain transactions in token. To do this, the contract tracks the amount of gas spent in a transaction, computes the price paid for it in ether and sends it back to the user, after deducting the equivalent token amount from their account. The contract continuously maintains an ether balance, which it uses to pay for the refunds and which it refuels through a public auction for token as soon as it falls below a specified limit.

### 3.2 Web development and Ethereum: Web 3.0

Traditionally, users of the web interact with a central server hosted on providers like AWS or Heroku through a browser that executes code written in HTML for structure, CSS for design and Javascript for function. Browsers can request data from servers by performing HTTP GET requests, to which the server will respond by sending back the requested data if the request is legal. To save data to the server, browsers can perform an HTTP POST request. Requests can be made synchronously, where the execution of client-side software waits for the request to be fulfilled, or asynchronously, where the program can continue executing while waiting for the response of the server. Most requests today are performed asynchronously to improve the user experience. The next commonly used requests are HTTP PUT to update and HTTP DELETE to delete data. With ethereum, this server is stored in the form of a blockchain that is maintained by

every node in the network. The state of the blockchain is agreed upon by using the previously mentioned consensus algorithms. Users can connect to the blockchain node using the web3.js library, which in turn communicates with the blockchain through remote procedure calls (RPC).

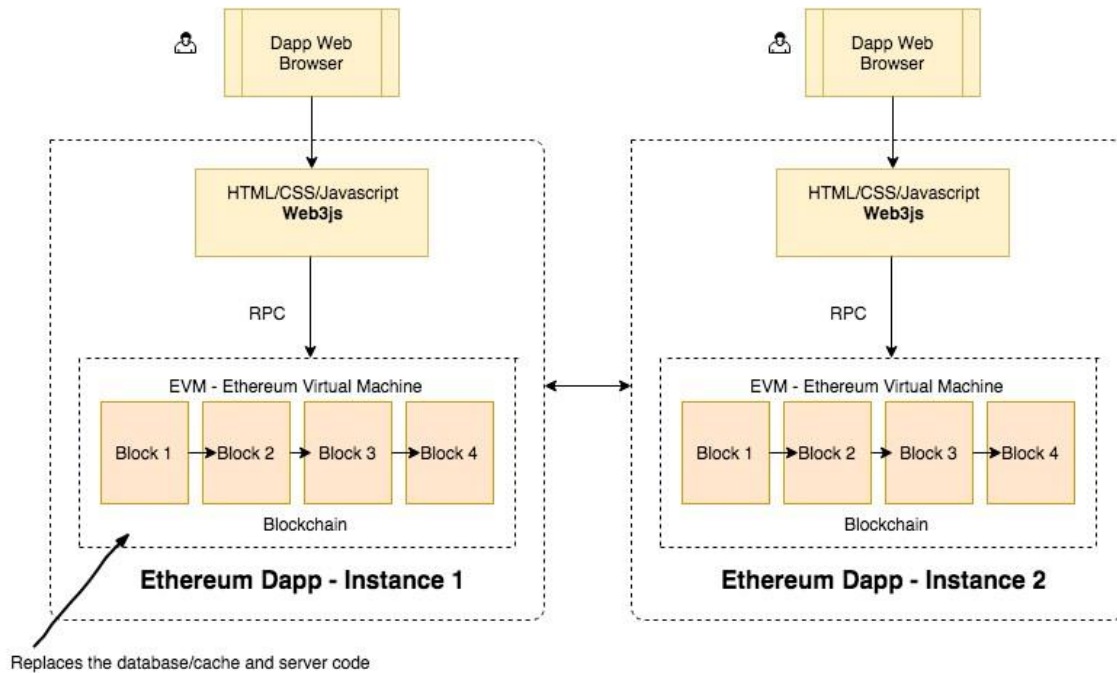


Figure 2, Web design using Ethereum[15]

A decentralized application can be deployed to the Ethereum network through the following steps:

1. Write smart contracts in Solidity, or another Ethereum language like Viper, and compile the source code to bytecode.
2. Start an Ethereum node
3. Deploy the contracts to the Ethereum network. This is done by the running node, which signs the transaction using the default account if none other is specified. Enough gas must be supplied to the transaction for successful deployment. Once the contract is successfully mined into the network, a transaction receipt is returned that contains meta-data required for future interaction with the contract, like its address and Application Binary Interface, which is a list of available variables, methods and events that can be called in a contract.
4. Clients can interact with the contract by supplying the address and ABI in JSON format to the web3 contract object, which will send calls to the ethereum network and receive return values. All successful transactions will be propagated through the network, while it is also possible to call functions that don't change the state of a contract locally without having to pay for it in gas. Such functions are identified with the keyword *constant* and any changes made to a contract during the execution of a *constant* function will not be recorded in the blockchain and therefore not persist.

Combining the traditional web architecture with the web 3.0 is a powerful technique to host a site on a traditional server, while also having access to the benefits of blockchain technology, as is elaborated in the next section. To avoid having to run an eEthereum node on the server, we use



MetaMask, which provides an extension for the Chrome browser and lets users interact with the ethereum blockchain straightaway. It works by running a node to keep in sync with the blockchain and injecting a web3.js object into the Javascript environment of the browser. This makes MetaMask a single point of failure and ideally it should be configured to point to a locally run Ethereum node instead of its own node. The web3.js object is simply a library that provides functions that make interactions with an Ethereum node more straight forward. MetaMask will be discussed in more detail in the implementation section the Xcoin architecture is summarised in Figure 3.

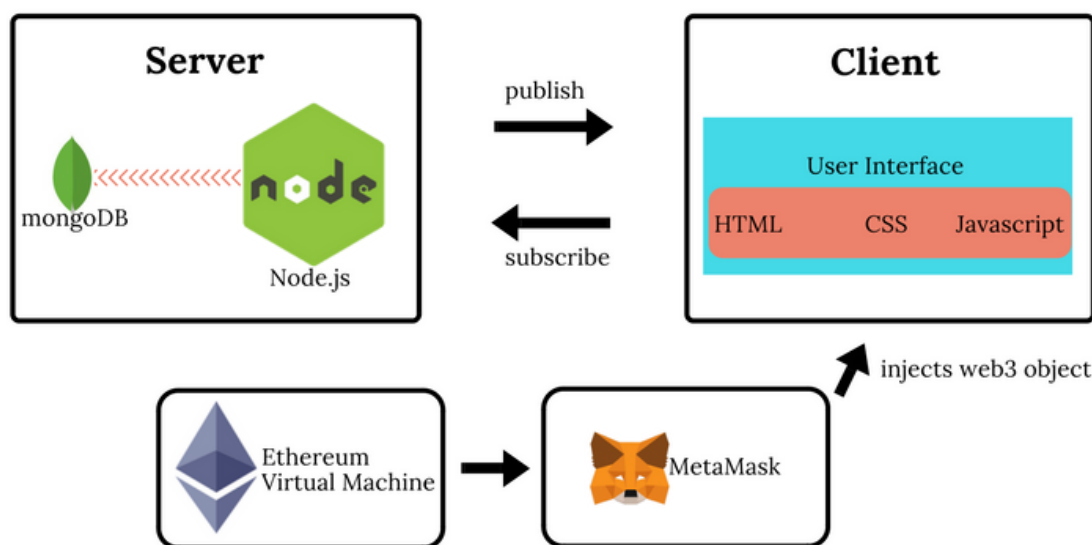


Figure 3, Xcoin architecture

### 3.3 Databases

Storage on the Ethereum network is expensive, and attempting to store all the data that is needed for a forum in it will prove expensive. The advantage of using the Ethereum blockchain is that data is secured within the ledger, making it difficult for anyone to alter. To make use of this benefit while avoiding large gas costs, a system will be implemented in which the bulk of lesser important data is stored on a server database, while being secured with a hash key that is computed from its content and stored on the blockchain. An illustration of this system is shown in figure 4. To further secure data storage, only HTTP POST and HTTP GET requests are allowed, avoiding the need for a centralised system of authentication. Objects are only stored on the database if no previous object with the same ID exists. When data needs to be updated, a new object is created that points to the last version. Furthermore, to prevent an attacker from precomputing the ID of an object and preventing other users from posting data, a timestamp is included in the hash as a source of unpredictability. Corrupt data will therefore be detected, but cannot be recovered, for which important data like the address of a contract is stored entirely on the blockchain. A hash is one way cryptographic function that produces a fixed size output from a variable sized input. It is basically impossible to compute the input given the output, but computing the output from the input is fast and simple.



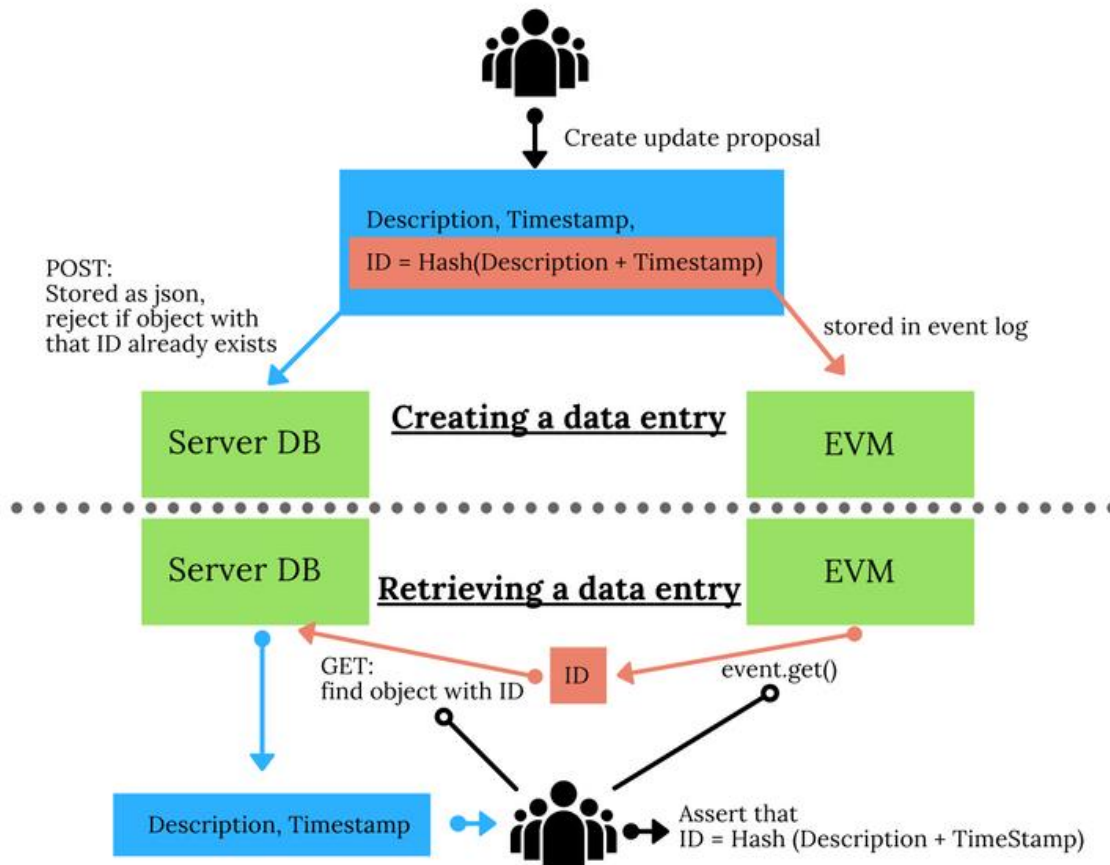


Figure 4, Hybrid data storage, using Web2.0 and Web3.0 architecture together

As Node.js is used to implement server-side execution of the application, MongoDB is a convenient choice of database as they both use Javascript and JSON. MongoDB is a NoSQL database that stores information in JSON-like objects, instead of rows and columns like traditional databases. Data that will be stored in mongoDB includes comments, user information, descriptions of bugs and update outlines for the developer, none of which is sensitive information. Issues of privacy are therefore not applicable and the only need for privacy is in the storage of a user's password, which is done using the passport middleware, as is explained later.

### 3.4 Events

Xcoin relies heavily on events to facilitate communication from smart contracts to the front-end user interface. In traditional web development, a server response is provided in callback to the front-end. In the Ethereum network, every time a transaction is mined into a block, the executing smart contracts can fire events that are stored on the blockchain and can be processed by front-end applications. Events are used in two crucial ways:

1. To return values from smart contracts

As discussed earlier, *constant* functions in contracts can be called from the user interface without having to pay gas for them as they are not included in the blockchain. This also means that a return value can be obtained immediately as the application does not have to wait for the function call to be mined into a block. Calling a *constant* function is referred to as a call, while executing a function that needs to be included in the blockchain due to changing its state is called a transaction. When a transaction is executed, the immediate return value that the front-end receives is the transaction id, not the return value. To obtain the return value, one must wait for an event to fire and pass back the value when the transaction is mined. Other code can continue to execute while the event is waited for.

#### *Code snippet 1, events in js*

```
//In smart contract

Contract example {

    event VoteHasEnded(uint256 voteId, uint256 time, bool successful);

    function endVote() {

        VoteHasEnded(voteId, now, true);

    }

}

//In front-end

var voteHasEnded = contractInstance.VoteHasEnded({value: filter.for.this.value});

voteHasEnded.watch(function(error, result) {

    console.log("Vote ID:" + result.args.voteId);
    console.log("Has ended at :" + result.args.time);
    if (result.args.successful) {
        console.log("And was successful");
    } else {
        console.log("And was unsuccessful");
    }

});
```

## 2. As a cheaper form of blockchain storage

Technically, events are stored as logs, which is also what they are referred to in the Ethereum Yellow Paper. Logs were designed to be a form of storage that is significantly cheaper than contract storage. A log costs 8 gas per byte, whereas a contract storage costs 20,000 gas per 32 bytes, which makes a nearly hundred fold difference. Although logs are not accessible from within contracts, even if they fired them, they form the perfect tool to create a history of the executions by the contract in a front-end environment. To get all past events, one can use *event.get()* instead of *event.watch()*.

### 3.4 Smart Contract Design

Since Solidity was chosen as language to implement the smart contracts, we have access to its object-oriented capabilities and the built in contract class. When users want to make a new contract, all that is therefore needed is to create a new instance of the given class.

Three contracts are designed for the operations of Xcoin:

#### 1. *TokenCreator*

This contract is the first and only contract that is deployed to the blockchain by the creator of Xcoin. It's only function is *makeTokenManager()*, which can be called by anyone to create a new *TokenManager*.

#### 2. *TokenManager*

This contract is deployed to the blockchain through the *TokenCreator* by users of Xcoin. One *TokenManager* exists for every token and it cannot be updated. This contract resides over voting, bug hunts, bounty hunts, auctions and it ensures smooth transitions to updated tokens. It also tracks the address of *Token*. Client side software can monitor processes from within this contract by receiving events. When a *TokenManager* is made by a user, it also creates the corresponding default *Token*, setting its parameters to the specified values.

#### 3. *Token*

Is created and updated by *TokenManager*. Can be replaced by updated versions. Should allow facilitate methods specified in ERC20 standard to facilitate standard token procedures.

The *Token* contract includes data fields and functions, representing its state and state transitions respectively. To be compatible with most wallet interfaces, the *Token* needs to implement a set of functions outlined in the ERC20 standard shown in Figure 5. To implement a token, a contract is created that contains a mapping of public addresses to balances; this is the state. In addition, functions need to be added to allow for transfer of funds, mining of the token and other functions.

To customize a token, a user specifies the aforementioned parameters in a web-interface and the contract code is instantiated with those values. An update to a token is implemented by deploying a new contract which adds new functions or modifies already existing ones. E.g. *.transfer()* can be modified to call a second function *giveToCharity()*, which takes a percentage of the transactions and transfers it to an account owned by a given charity. When a token is updated, a new contract is created, and the *TokenManager* copies the state of the previous token to the new token and updates the address.

The *TokenManager* also contains meta-data representing current state and functions representing state transitions. Although the contract has no access to its past states, snapshots of it can be stored on the blockchain using event logs and a history of states can therefore easily be rebuilt and analysed. This is important in the case of disputes between stakeholders of the contract

operations. Using event logs means that the contract is freed from having to store its own history and saves storage. If the contract ever needed to access information from one of its past states, it can ask a web-client to forward this information, although this should rarely be required.

```
1 // https://github.com/ethereum/EIPs/issues/20
2 contract ERC20 {
3     function totalSupply() constant returns (uint totalSupply);
4     function balanceOf(address _owner) constant returns (uint balance);
5     function transfer(address _to, uint _value) returns (bool success);
6     function transferFrom(address _from, address _to, uint _value) returns (bool success);
7     function approve(address _spender, uint _value) returns (bool success);
8     function allowance(address _owner, address _spender) constant returns (uint remaining);
9     event Transfer(address indexed _from, address indexed _to, uint _value);
10    event Approval(address indexed _owner, address indexed _spender, uint _value);
11 }
```

Figure 5, the ERC20 Token Standard[16]

## 3.5 Smart contract best practices

The following guidelines [17] are taken from a document published by ConsenSys, a company supporting the development of decentralized applications, which summarizes a set of smart contract best practices to avoid bugs. Only guidelines relevant to Xcoin are mentioned. Bugs should be avoided at all costs, as source code cannot be changed once it has been deployed to the blockchain:

### 3.5.1 Send() vs. transfer() vs. call.value()

Send(), transfer() and call.value() are all functions that can be used to transfer ether to another account. When a contract receives ether, its fallback function is triggered, which is a function that takes no arguments, returns no values and contains the identifier *payable*. If a contract does not contain a fallback function and receives ether, an error will be thrown. With the three different functions to send ether come different benefits and dangers. Call.value() can be sent with any amount of gas under the maximum limit, which means that the receiving contract has the option to perform additional functions when receiving ether. An attacker could exploit this to re-enter the calling function before it has finished, as flow control is now in their hands. To avoid this problem, send() and transfer() are both sent with a fixed amount of 2,300 gas, which is only enough to log an event. With the gain in security comes a loss in flexibility as the called contract cannot implement additional features upon receiving ether. The difference between transfer() and send() is that the former throws an error if the funds could not be transferred, and send() only returns false. This means that an attacker can cause the contract to continuously halt execution when using transfer(), and that send() needs to be checked after each call to confirm successful fund transfer. A fourth solution that has become a common practice is to let the recipients of funds withdraw them independently. This solution will be adopted for Xcoin, as it is deemed the safest option and simplifies the logic in the contracts.

### Code snippet 2, fallback function

```
// fallback function
Function() payable {

    //can max trigger one event for <address>.send() and <address>.transfer()

    //can do more with <address>.call.value()
    //..
    //like perform an attack
}
```

## 3.5.2 Race conditions

A race occurs when function A in contract A calls a second contract B, which then calls the original function A again. Alternatively contract B could call another function B from contract A that acts on the same state variables as function A. The danger is that variables can be modified asynchronously and therefore give incorrect results: the second instance of function A changes state variable X before the first instance of function A does, therefore giving an incorrect result. To fix this, always make external calls the last step of a function. If this is not possible, mutex locks can be used, which allows you to lock some state so that it can only be modified by the owner of that lock. However, care has to be taken that no-one can lock a mutex indefinitely or create deadlocks, for which reason they will be avoided in Xcoin and instead external function calls will always be placed at the end of a function.

## 3.5.3 Upgrading a faulty contract

Updates to the *Token* contract are decided through voting and reaching a consensus could take too long to implement a quick upgrade in an emergency. A circuit breaker is one possible solution although it opens the contract to new vulnerabilities in which circuit breakers are triggered by an attacker when operating normally. A circuit breaker can either be triggered automatically by the contract once certain programmable conditions are fulfilled, control can be given to trusted parties or a combination of the two can be used.

## 3.5.4 Timestamp dependence

Miners have the power to manipulate the timestamp of a block and all direct and indirect uses of timestamps have to be considered. A possible solution includes estimating the time by using *block numbers* and *average block time*, but this is not a viable solution in the long-term as block times may change, as is expected in Casper, the next release of the Ethereum protocol. Another possible solution is to create an event that fires and transmits the timestamp to a front-end, that it compares it to its own clock and trigger security functions in the contract when the timestamp has been altered.

#### Code snippet 3, timestamp dependence

```
uint256 someVariable = now + (5 * minutes);

if (now > votingDeadline) { //the now can be manipulated by the minter
    //implement update
}

If (someVariable % 2 == 0) { //someVariable can be manipulated by miner
    //Do something
}
```

### 3.5.5 General good practices

- Ether can be forcibly sent to an account address, so avoid checking account balances for a fixed value
- Keep fallback functions simple
- All data on the blockchain is public, use hashing for sensitive information
- Explicitly mark visibility in functions and state variables:
  - *private*: only visible from within contract
  - *internal*: only visible from within contract and derived contracts
  - *public*: visible to all
  - *external*: similar to public, but when called from within contract needs keyword *this*

### 3.5.6 Error handling

Similar to other languages, errors can be ‘thrown’ based on certain conditions. However, unlike other languages, the error cannot be caught. This is because solidity performs a revert operation (opcode 0xfd) when encountering a user-defined exception like *require(false)* or *throw*. When encountering run-time exceptions like *Out of Gas* or *assert(false)*, it performs an invalid operation (opcode 0xfe). In both cases, all changes are reverted back to the state at the beginning of the call and all gas is consumed. Instead of *require()* or *throw*, conditional statements can be used instead to check for correct user inputs and exit the function upon illegal input, which saves the user from spending all their gas. User-defined exceptions should only be triggered when reverting the contract state is explicitly desired.

#### Code snippet 4, error checking

```
// throw user-defined exception
If (true) {
    throw;
}

// or
require(false);

// throw run-time exception
assert(false);
```

### 3.6.6 Summary

To summarize, the expected functions of the applications are:

1. User can create new TokenManager and Token with customizable features
2. User can choose already existing Token and enter the corresponding forum with a username and password to prevent spamming
3. In Forum:
  - a. Community is informed about latest events with clear instructions on how to participate
  - b. Community can create a new topic for an update idea they have
  - c. Community can to start a bountyHunt for any existing update topic
  - d. Community can receive an overview of the Token attributes
  - e. Community can use all public methods of the Token contract including newly added features from updated Token
  - f. Developer can bid for bounty
  - g. Developer can submit update implementation and get paid for it
  - h. Community can find bugs in update implementation submitted by developer
  - i. Community can vote on legitimacy of found bugs
  - j. Community can bid in token auction
  - k. Community can choose to pay for transactions in token instead of ether
4. Smart Contracts can:
  - a. Token
    - i. Facilitate mining of token.
    - ii. Implement methods of ERC20 standard
  - b. TokenManager
    - i. Maintain the logic behind all activities that community and developer take part in
    - ii. Send regular information to front-end about current state
    - iii. Process outcome of activities mentioned in 3. and perform required response
    - iv. Facilitate transition between old contract Token to updated Token
  - c. TokenCreator
    - i. Create new TokenManagers and Tokens
5. Maintain a hybrid data storage system, combining benefits of server databases and blockchains

## 4: Implementation of Prototype

### 4.1 Development Process

#### 4.1.1 Ethereum clients

The vast majority of the development process was not done against a live blockchain, but on a private network through a client called TestRPC.

“testrpc is a Node.js based Ethereum client for testing and development. It uses ethereumjs to simulate full client behavior and make developing Ethereum applications much faster. It also includes all popular RPC functions and features (like events) and can be run deterministically to make development a breeze.”[18]

When TestRPC is started, it creates 10 accounts preloaded with 100 fake ether each, and a new block is mined every time a new transaction is sent to the network. This makes development extremely fast. One weakness of the TestRPC client is the built-in block explorer that seems to contain bugs. As an example, when calling a function that creates a contract, which in turn creates another contract, the explorer will only signal the creation of one contract instead of two. As a workaround to confirm deployment of the second contract, the first contract returns the address of the second contract, and a test transaction is sent to it to check its existence.

To deploy a contract to the private network, these steps are followed:

1. Run testRPC on localhost:8545
2. Enter the node console and initialize the web3 object to connect to localhost:8545
3. Compile the contract into EVM bytecode using the Solidity compiler
4. Create a contract object, parsing it the ABI from the previously compiled contract
5. Deploy the contract to the private network, parsing it data for the constructor, together with the compiled code, the gas limit and the account that you send it from

The equivalent, executed on the command line is:

```
1. >>./testrpc (listens on localhost:8545 by default)
2. >>node
   >>Web3 = require('web3')
   >>web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
3. >>code = fs.readFileSync("Example.sol").toString()
   >>contract = web3.eth.compile.solidity(code)
4. >>ExampleContract = web3.eth.contrac(contract.info.abiDefinition)
5. >>deployedContract = ExampleContract.new(<constructorData>, {data:
   contract.code, from: web3.eth.accounts[0], gas:4700000})
```

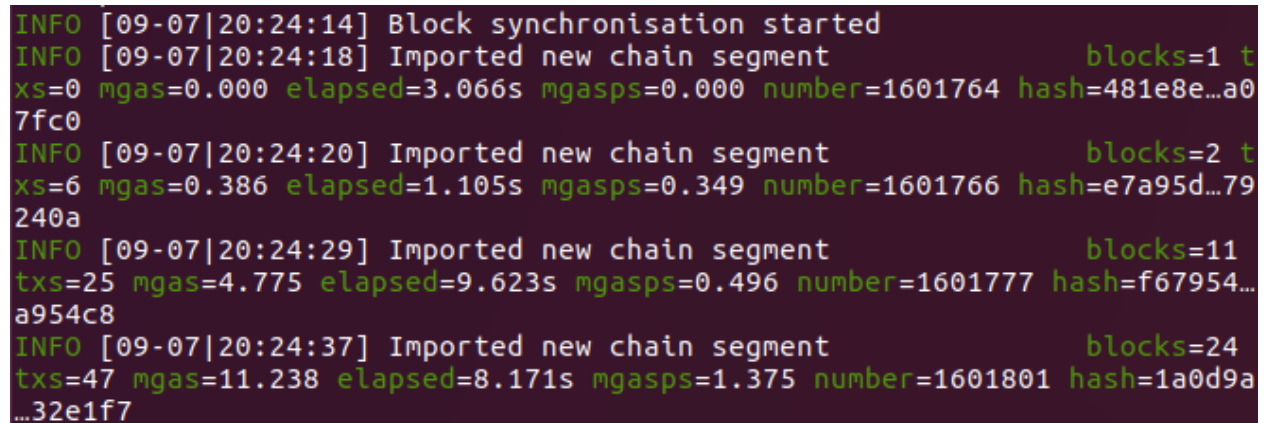
To deploy contracts to the Ropsten test-network, which is much more similar to the mainnet than the private network in TestRPC and therefore required for testing, a second client called geth is used.



Geth is one of the most popular client implementations in the Ethereum community and is entirely run from the command line. Written in Go, it is the main deliverable of the Ethereum Frontier release. To start a geth node and synchronize it to the Ropsten test-network, the following command is entered at the command line:

```
>>geth -testnet -syncmode "fast" -rpc
```

This will start the node, automatically connect to other peer nodes and start downloading, or 'syncing', the blockchain, a process that can take up to two days.

A terminal window with a dark background and light-colored text showing the Geth node's progress in synchronizing the testnet blockchain. The output consists of several log lines indicating the start of block synchronization and the subsequent import of new chain segments. Each segment's details, including transaction count (txs), gas used (mgas), elapsed time (elapsed), gas price (mgasps), block number (number), and hash, are displayed. The process shows the node successfully importing multiple segments, with the block number increasing from 1601764 to 1601801.

```
INFO [09-07|20:24:14] Block synchronisation started
INFO [09-07|20:24:18] Imported new chain segment          blocks=1 t
xs=0 mgas=0.000 elapsed=3.066s mgasps=0.000 number=1601764 hash=481e8e...a0
7fc0
INFO [09-07|20:24:20] Imported new chain segment          blocks=2 t
xs=6 mgas=0.386 elapsed=1.105s mgasps=0.349 number=1601766 hash=e7a95d...79
240a
INFO [09-07|20:24:29] Imported new chain segment          blocks=11
txs=25 mgas=4.775 elapsed=9.623s mgasps=0.496 number=1601777 hash=f67954...
a954c8
INFO [09-07|20:24:37] Imported new chain segment          blocks=24
txs=47 mgas=11.238 elapsed=8.171s mgasps=1.375 number=1601801 hash=1a0d9a
...32e1f7
```

Figure 6, geth node downloading the testnet blockchain

## 4.1.2 Frameworks and Tools

### Truffle

Truffle has become the most popular framework for Dapp development and comes with built-in smart contract compilation, linking, deployment and binary management. Originally built for client side applications, the directory structure had to be modified to include the server-side architecture of Node.js using embedded Javascript.

### Node.js and Express

To run the back-end engine, Node.js and Express were chosen as they are extremely popular choices of frameworks and are common in many software stacks. This gives access to a wide variety of examples and tutorials that could be used to receive a quick introduction into back-end web development. Node.js is lightweight and efficient Javascript runtime built on Chrome's V8 Javascript engine. It was created in 2009 to allow developers to use JavaScript both as a back-end and as a front-end language. It is single threaded and uses non-blocking I/O calls, which allows it to support thousands of concurrent connections without the cost of thread context switching, in which states of processes must be stored and restored to allow multiple processes to share a CPU at the same time. Its package manager, npm, is the largest collection of open source libraries in the world and was very useful to include more than 15 different libraries, of which the most important were:

- Mongoose:  
Allows for straightforward, schema-based modelling of data stored in mongoDB.  
Includes a useful feature with which data can be formatted before being stored that is used to format the contract ABI that an update developer submits with the

implementation of the update. Mongoose could guarantee that it was stored in a format that was later readable by the web3 contract object.

- Express:  
Provides an API for HTTP utility methods and middleware addition. Middleware is software through which HTTP requests are passed first before reaching the requested file. An example of a middleware implementation in this project is user authentication.
- Passport:  
Is used for user authentication by implementing it as middleware. One can selectively choose which routes should be redirected to the passport middleware and which should be displayed without authentication. This makes it easy to organise the webpage into pages that are subject to authentication and pages that are available to the public. Passport uses another library called BCrypt to only store the SHA256 hash of the password in the mongoDB database to prevent an attacker from retrieving the original password in the event of a successful attack.
- Web3:  
This library provides the web3 object, which communicates with the local Ethereum node through RPC calls and can interact with contracts deployed to the blockchain. For a full documentation of available functions, see the official API reference.[19]
- Webpack:  
Webpack is a module bundler that is extremely convenient for compiling JavaScript files that import other modules. In its configuration file, different loaders can be specified to compile different file types, such as a json-loader for JSON files and the babel-loader for JavaScript files. This way any type of file can be imported into a JavaScript file in the right format, making development modular and flexible.

### EJS, JavaScript, HTML and Bootstrap

To implement the front-end of the site, it was initially intended to use the framework AngularJS due to its vast range of capabilities. However, it quickly became clear that its learning curve was too steep and that it offered more complexity than was required for this project, and that using the CSS framework Bootstrap in combination with Embedded JavaScript (EJS), HTML and plain JavaScript would be sufficient to showcase the capabilities of the back-end engine, even if the UI is not as aesthetically pleasing as it would have been with AngularJS. HTML is used for the structure of a webpage, while Bootstrap is used for its design, providing aesthetically pleasing designs, JavaScript for the dynamic adjustment of webpage content depending on user input and EJS as a rendering service that fills html pages with data before sending it to the browser.

Combining truffle, node.js and webpack, the steps for prototyping the Dapp were:

1. Start the TestRPC client
2. Compiling and deploying the smart contracts to the private network, by calling 'truffle migrate'
3. Starting the mongoDB database by calling 'mongod'
4. Importing different modules into JavaScript files by calling 'webpack'
5. Starting the server by running 'node server.js' or 'nodemon server.js'

After these five steps, the application is available on localhost:8080

## 4.2 Solidity Contracts

### 4.2.1 Final Architecture

The implementation of the contracts followed the design specifications closely and the functions remained as they were initially intended. Instead of having a single *Token* contract, however, it was decided for the sake of clarity, that it should be split into three parts. The first is *ERC20Standard.sol*, which is an abstract interface with variables and methods that are required by the ERC20 standard. The second is called *BaseToken.sol*, and includes all function implementations for the ERC20 standard and all variables and methods that are required by the *TokenManager* contract. The third is *NewToken.sol*, which only contains a constructor and is the contract where updates should be implemented in. Any update to *NewToken.sol* must inherit from *BaseToken.sol* and both *BaseToken.sol* and *ERC20Standard.sol* should be left untouched. This is a neat way of ensuring that new updates are still compatible with *TokenManager.sol*. The contract class of *NewToken.sol* that is included in *TokenManager.sol* will work with new updates as long as all previous methods and variables are still available, even if new methods are added to the contract that are not part of that class. To clarify this structure, the corresponding UML diagram without variables and methods is shown in Figure 7.

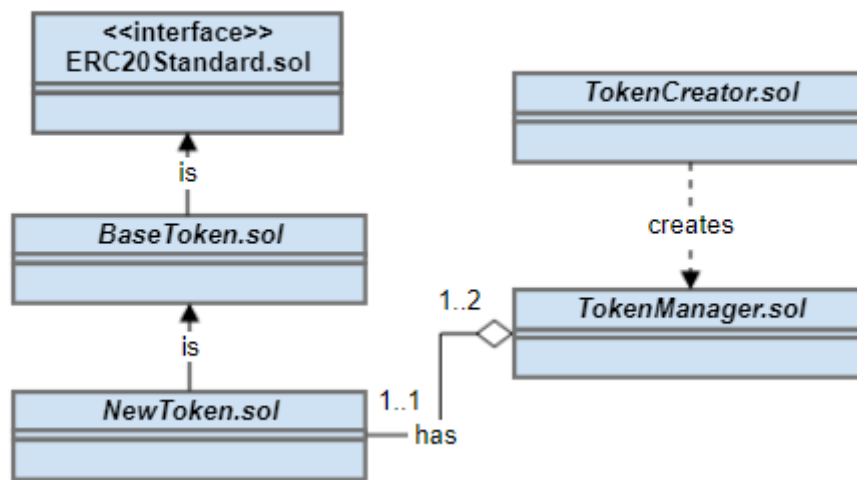


Figure 7, UML diagram of contract classes

The reason that *TokenManager.sol* can have two instances of *NewToken.sol* is the transition phase between accepting a new update and implementing it, where *TokenManager.sol* keeps track of the old and the new contract before killing the old one. A more functional representation of the interactions between the contracts is shown in Figure 8.

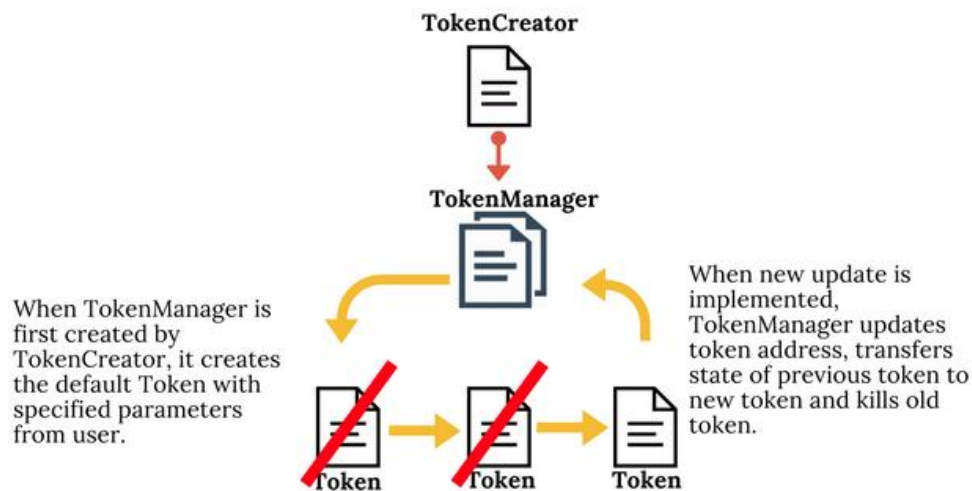


Figure 8, Functional relationships between contracts

The *TokenManager.sol* stays the same throughout the lifetime of token, guaranteeing the reliable execution of procedures that are essential to the organisation of the community, like votes and update implementation. *TokenManager.sol* is a relatively large contract and instead of having separate contracts for votes, auctions, bounty hunts, bug hunts and update implementation, it was deliberately decided to have all functions inside one contract, to reduce costs of its execution and to enhance the synchronicity of its state transfers.

One of the initial specifications of customizable features in a token included mining consensus algorithms. If the token lives on the Ethereum blockchain, there is no need for mining as the funds are already secured by the ether miners. Instead the tokens are simply issued according to a linear equation that the founder specifies at creation of the token, and are added to the balance of the *TokenManager*. To use these funds, a function *mint()* is included in *BaseToken.sol()* that allows only the creator to release these funds into any account. Ideally, the creator would specify the linear equation such that tokens only start being released after a certain amount of time, which gives him and the community time to talk about ways to change that function and implement other methods on how funds can be released to the public.

## 4.2.2 Contracts in detail

### *TokenCreator.sol*

As outlined in the design section, this contract only has method, which is to create a new instance of *TokenManager.sol*. Although it does not keep a registry of the contracts it created, it fires an event after creation including the address of the token manager, the address of the token, and other data about the creation. These events form the entry point into the whole application, as the first thing a user does before entering a forum is to get a list of triggers of this event. Based on this list, they can then enter a forum by using as URI the hash of the manager address, the token address and the creation time.

## *TokenManager.sol*

This contract forms the core of the application. With over 600 lines of code, it is on the limit of contract sizes, as any larger size will cause the gas cost of its creation to surpass the transaction gas limit, although these limits are increasing. The processes that the *TokenManager* controls are organised into Structs, and are the following:

- BountyHunt
  - Can be started by anyone
  - Can only be started if no other BountyHunt, Update or ChangeOver is active
  - Fires an event with the hash of the description, so developer and community have a record of the 'contract' they agreed upon.
  - Anyone can bid for bounty with a fixed price/hour ratio, meaning that the winner is the one to offer an implementation in the shortest time period. Hourly wage stays the same.
  - Once the BountyHunt deadline has been reached, no more bidding is allowed and anyone can end it, triggering the next step of the update implementation lifecycle.
- Update
  - This phase describes the time period from the moment the developer starts working on the implementation to the moment the update is accepted or rejected.
  - Within its time frame, the BugHunt is included as the developer might have to correct bugs after the BugHunt.
  - This time period can only be started by the *TokenManager* contract, which it does automatically after the community approves of the update through a vote.
  - If developer misses the deadline, he cannot submit the update anymore and anyone can end it.
  - If developer finishes on time, only they can submit the update with the address of the newly deployed contract. On the front-end they also upload the contracts source code and ABI. Once submitted, the *TokenManager* automatically starts a BugHunt.
  - If no bugs are found, the developer is paid, and the new contract is initialised to the state of the old contract
  - Although funds are secured in a contract, the developer is guaranteed payment. It is similar to a central bank saying "we will pay you from our reserves, and if we don't have any reserves left, we will print more money for you", only that this promise is hard coded into the smart contract which is immutable as it lives on the blockchain
  - If a bug is found, the *TokenManager* checks if the developer has any remaining attempts and resets the new deadline for fixing the bug.
- BugHunt
  - Describes the period in which the address of the new contract and its source code are publicized for review by the public to find bugs. Anyone can compile the source code and check that it matches the bytecode stored at the address.
  - Only one bug can be found at a time

- Finder of bug submits a description of it whose hash is stored on the blockchain
  - After a bug is found, the community votes on whether they think this is a bug or not. This step is required as anyone can find a bug, not just qualified professionals.
  - To avoid spamming, a bug finder must pay a small deposit of the equivalent of \$0.50 in ether.
  - If the bug is verified by the public vote, bug finder gets paid a set percentage of the updater's salary. This stops the developer and the bug finder to cooperate to make more money.
  - If the bug is not validated, the BugHunt continues
- Vote
    - Can only be started by *TokenManager*, either after a bug is submitted or after a BountHunt finishes.
    - Freezes the balance of the *TokenManager* and calculates the amount of votes needed for the success or failure of the vote, depending on how much token is in circulation and the consensus required for successful votes.
    - Once a user votes, their savings are frozen and the vote counted as proportional to their account balance.
    - The vote finishes either by passing the deadline, in which case anyone can end it, or by fulfilling the requirements for a successful or unsuccessful vote prematurely. Depending on whether the vote was on an update or a bug, the contract automatically starts the next phase of the update implementation lifecycle.
- ChangeOver
    - This period starts when a new contract is accepted as bug free and ends when the old contract is killed
    - As it would be expensive to keep a registry of all accounts and their balances, the *TokenManager* cannot transfer everyone's funds automatically to the new contract. Instead users are given this time window to call the function *transferToNewContract()*
    - When the period finishes, anyone can end the period and cause the *TokenManager* to kill the old contract, transferring the recovered ether to its own balance.
- Auction
    - When the *TokenManager* runs out of ether after refunding transactions to community members. A target is set by the community on how much ether the contract should refuel to and the contract estimates the equivalent amount of token. It then offers this amount of token in a public auction where anyone can acquire it by bidding for it in ether.
    - To guarantee that bidders will actually pay the price, the funds are sent to the *TokenManager* together with the bid. In case they get overbid, the previous bid is

added to list of returned bids, from where the owners can withdraw it independently.

#### *Code snippet 5, system of withdrawing payments instead of sending*

```
function bid() payable {  
  
    //throws if someone tries to bid before or after the bid, or if bid  
    //is too low.  
    require(now >= auction.startTime && now <= auction.endTime  
    && msg.value > auction.highestBid);  
  
    If (auction.highestBidder != 0) {  
  
        //beaten bid is added to withdrawable bids instead of being  
        //sent back to the owner directly  
        auction.returnBids[auction.highestBidder] += auction.highestBid;  
    }  
    auction.highestBidder = msg.sender;  
    auction.highestBid = msg.value;  
  
    //event for front-end  
    NewHighestBid(auction.id, msg.sender, ms.value);  
}
```

A pattern emerges in which the *TokenManager* dictates **what** should be done, but relies on users to activate these actions at the correct times. An alternative was considered in which the *TokenManager* makes an external call to the existing Alarm Clock contract [20] that works as a timer and notifies the *TokenManager* when the time window it specified is over. So far the *TokenManager* makes no external function calls, greatly enhancing its security and the cost of compromising security outweighs the benefits gained from a timer contract. In general, there is usually at least one person that should have incentive enough to send these activation signals to the *TokenManager*. When ending a successful update vote, it is the developer and when ending a successful bug vote it is the bug finder. To ease development, the *TokenManager* throws an error when users try to do something illegal. A more generous approach of using conditional statements without throwing errors will be implemented in future versions to stop users from losing unnecessary gas. Currently, if two users make the same bid for the auction at the same time, one of them will cause the contract to throw an error and lose all the gas they sent with the transaction, which is unfair to the user.

#### *BaseToken.sol*

Not only is this contract in charge implementing the ERC20 standard functions, but it also contains functions that are needed by the manager and can only be called by the manager, like *block()*, which freezes an account, *empty()*, which empties an account, or *initialise()* which initialises a new update contract to the state of the previous contract. *BaseToken.sol* is also in charge of issuing the token, based on the linear equation supplied at creation time. A passive technique is used to implement this feature, where the contract only updates the supply of token when a user makes the first fund transfer on a given day. The contract keeps track of how many days have passed since the last fund transfer. In other words, the total token supply is only an



abstract number until the point where somebody makes a transfer and the balance of the *TokenManager* is updated to include the newly issued tokens. All functions that need to update the token supply before execution include the modifier *updateSupply()*, of which an outline is shown in code snippet 6.

*Code snippet 6, updateSupply() modifier*

```
modifier updateSupply() {  
  
    //checks if limit of issuance is already reached (upper cap or zero)  
    If (!limitReached){  
  
        //checks if supply is up to date  
        If (daysPassed > lastUpdate){  
  
            //update supply according to linear equation  
            //tokensIssuedToday = m * (daysSinceTokenCreation) + b  
            //where m and b are specified by creator of token  
            lastUpdate = today;  
        }  
    }  
    //continues with execution of function that called modifier  
    _;  
}
```

### *NewToken.sol*

This contract only contains a constructor for initial creation of the default token, and will be replaced by updated versions if communities manage to implement their ideas.

### *ERC20Standard.sol*

This contract only contains function declarations as it functions as interface for the ERC20 standard.

## 4.2.3 Testing

A big advantage of using truffle is that it comes with an automated testing suite. Tests can be written in JavaScript and in Solidity, of which JavaScript was chosen due to its longer history of performing tests. The test files are placed in the `./test` directory and can be executed using `'truffle test'`. The Mocha testing framework with Chai for assertions is used, which provides a familiar and enjoyable environment for writing unit tests. Instead of using the traditional *describe()* in Mocha, Truffle supplies a custom *contract()* function, which ensures that contracts are redeployed to the running Ethereum client before every new test to provide a clean contract state. *contract()* also supplies the test with a list of accounts that are made available by the Ethereum client running in the background. As the asynchronous calls are used to interact with a contract on the blockchain, a test is implemented as a chain of promises, of which each waits until the last transaction has been completed and included in a block. For rapid testing, the TestRPC client



was used for continuous testing during development, as there are no 15 second gaps between blocks. To ensure the functionality of the smart contracts on a more realistic network, tests were also conducted on the Ethereum Ropsten network using the geth client towards the end of the project. To simulate the time dependent outcomes of the methods, a simple function *wait(ms)* was used that would wait *ms* milliseconds before proceeding to the next promise. The tests used were extensive and a small part of one of them is shown in code snippet 8 to illustrate the concept.

*Code snippet 7, wait(ms) promise*

```
function wait(ms) {  
  return function(x) {  
    return new Promise (resolve => setTimeout(() => resolve(x), ms));  
  };  
}
```

*Code snippet 8, testing the contracts*

```
...  
    //start a bounty  
    managerInstance.startBounty(1,1);  
}).then(function() {  
    //make a bid for a bounty  
    managerInstance.bidForBounty(bountyBid);  
}).then(wait(7000)).then(function() {  
    //end bounty after waiting for deadline to pass  
    managerInstance.endBounty();  
}).then(function() {  
    //vote for update with 90% of tokens  
    return managerInstance.submitVote(true);  
  
//when passing the returned object of the previous function call,  
//an array of meta-data is passed that can be checked like  
//which events were fired with what values  
}then(function(result){  
  
    assert.equal(result.logs[0].event, 'NewVote', 'First event fired should be  
        NewVote');  
  
    assert.equal(result.logs[0].args.yes, 90, 'Should win vote with 90%');  
    assert.equal(result.logs[0].args.no, 0, 'No one should have voted against');  
    assert.isOk(result.logs[1].args.success, 'Vote should have succeeded');  
    assert.equal(result.logs[2].args.developer, accounts[0], 'Developer that won  
        bountyHunt and will implement update should be address in  
        accounts[0]: accounts[0]);  
...  
}
```

```
Contract: TokenManager
✓ Set parameters of tokenManager correctly (1792ms)
✓ Should issue tokens correctly; checked at different times (7332ms)
✓ Set up 5 second bountyhunt and vote; too little votes; fails (14479ms)
✓ Set up 5 second bountyhunt and vote; no developers; fails (7545ms)
✓ Set up successful 5 second bountyhunt and vote; developer fails to submit update (9849ms)
✓ Set up successful 5 second bountyhunt and vote; developer submits bug free code; update implemented (24186ms)
✓ IN UPDATE: Tokens should still be issued correctly, checked at different times (7859ms)
✓ IN UPDATE: Set up successful 5 second bountyhunt and vote, developer submits without bug in second try, update implemented (25101ms)
```

*Figure 9, Unit tests conducted in the truffle framework using the TestRPC client*

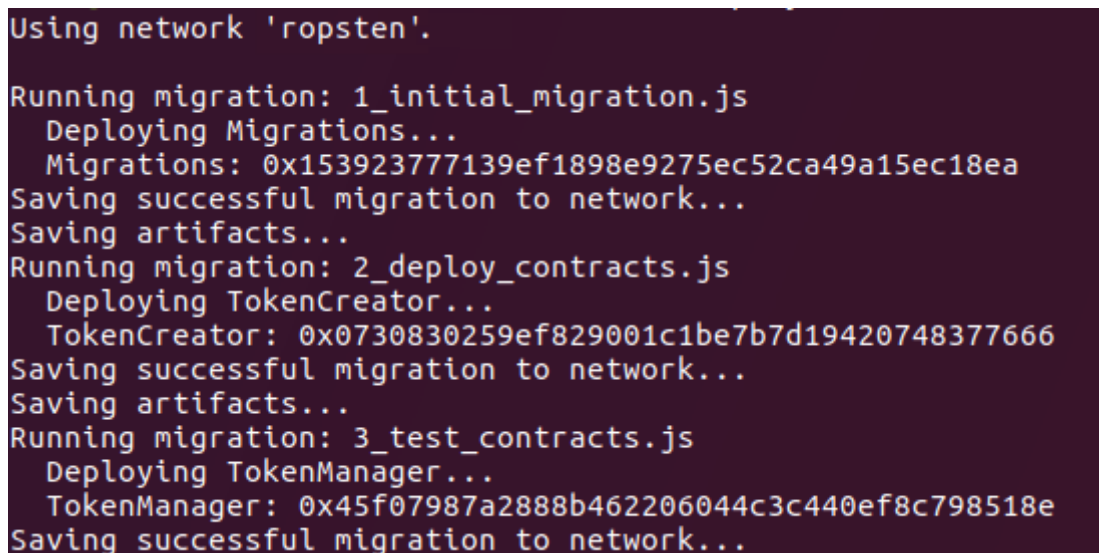
## 5: Deployment to the Ethereum network.

### 5.1 Networks

As described in section deployment, the geth client is started with the `--testnet` flag and once synchronized, the smart contracts can be deployed to the network through truffle. The testnet source code is identical to that of the mainnet, making it behave exactly like the mainnet in terms of logic. The only difference is that the vast majority of miners have decided to mine on the mainnet in order to earn real and valuable ether, instead of wasting their mining efforts on the testnet and that traffic on the mainnet is much higher. As miners usually choose to mine the transactions with the highest gas price first, larger traffic also leads to larger gas prices. This means that for the same transaction to be processed within the same amount of time on the testnet and on the mainnet, one will have to specify a larger gas price on the mainnet.

Once the contracts are deployed to the blockchain, a user can interact with them through the web3 JavaScript library. This project relies on a third party called MetaMask to inject the web3 object into the front-end Javascript environment. An alternative would be to run an ethereum node on the server itself, but it has been decided against that for the following reasons:

1. From the point of view of the user, both scenarios represent single points of failures, which means that he is vulnerable to an attack by or through either of the two platforms. The safest and most decentralized solution for a user is to run an Ethereum node themselves, although many won't have the necessary technical ambitions to do that today.
2. If a user decided to run their own node, MetaMask can easily be modified to point to the user's node instead of using the default node provided by MetaMask.
3. Running a node on the server requires computing power and is unnecessary given that tools like MetaMask exist.



```
Using network 'ropsten'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  Migrations: 0x153923777139ef1898e9275ec52ca49a15ec18ea
Saving successful migration to network...
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying TokenCreator...
  TokenCreator: 0x0730830259ef829001c1be7b7d19420748377666
Saving successful migration to network...
Saving artifacts...
Running migration: 3_test_contracts.js
  Deploying TokenManager...
  TokenManager: 0x45f07987a2888b462206044c3c440ef8c798518e
Saving successful migration to network...
```

*Figure 10, successful deployment of contracts to the ropsten test network using geth*

## 5.2 Metamask

MetaMask is an extension that users can download for the Google Chrome browser, which provides access to the Ethereum blockchain network through an easy to use UI, without the need of running a node. This is done by injecting a web3 Javascript object into each page that is loaded in the browser. Private keys are stored in a UTC key file, which can be decrypted using a user defined password. When a transaction needs signing, a MetaMask window pops up, asking the user to confirm the transaction and then signs it for the user.

To provide a wallet interface, MetaMask uses LightWallet, which is an HD wallet that stores private keys encrypted in the browser. It makes use of BIP32 and BIP39 [21] to create a tree of addresses using a randomly generated 12-word seed, or mnemonic, as seed. The wallet operates on a computer that is connected to the internet, giving it the name ‘hot wallet’. This makes it less secure than an offline wallet, because if a computer is connected to the internet, access could be gained remotely and keys can be stolen. Although the key files are encrypted, an attacker could gain access once a user has unlocked their MetaMask account, or by recording the password and gaining complete control over the machine. MetaMask and LightWallet are still in active development, and it is strongly discouraged to use them as long-term storage for ether. Instead, users should store the private keys in paper wallets or hardware wallets like the Ledger Nano S or TREZOR and only transfer limited amounts temporarily to the MetaMask wallet when needed. Future plans for MetaMask include providing users with an interface to use hardware wallets. As all tokens and ether on the ethereum network are sent by signing transactions with private keys, this applies to all currencies a user might hold.

In the MetaMask compatibility guide, a set of recommendation and requirements are set out for developing Dapps that use MetaMask [22]:

1. Check if the web3 object has been injected by confirming that the web3 type is not undefined. If web3 has not been injected, use a fallback strategy like a local node or failure. This check should be wrapped in a *window.addEventListener('load', ...)* handler to avoid race conditions with web3 injection timing
2. Because a user does not have a full blockchain stored on their machine, data lookups can be slow. MetaMask does therefore not support synchronous function calls. A synchronous function call means that the user’s interface is blocked until the function receives the requested return value.
3. The front-end should always check for the network that the current node is connected to by calling *web3.version.getNetwork()* and set the contract addresses of the underlying smart contracts appropriately

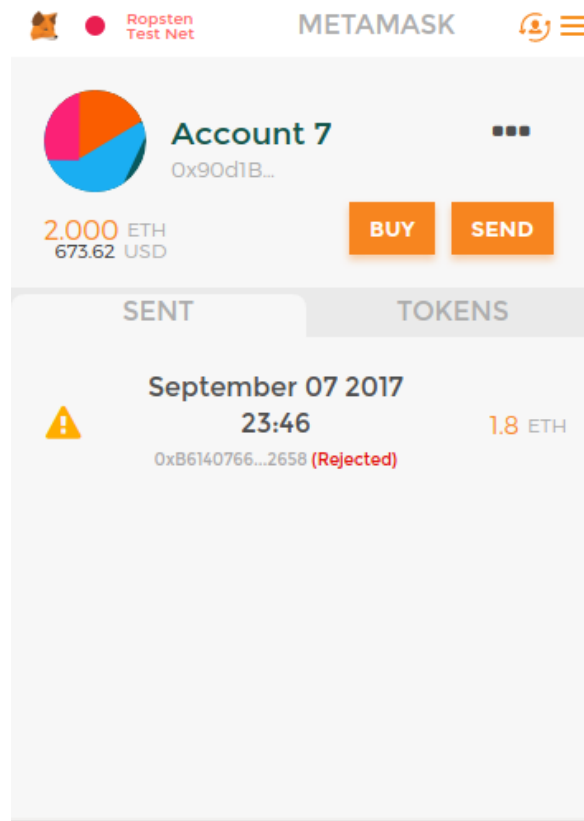


Figure 11, MetaMask UI

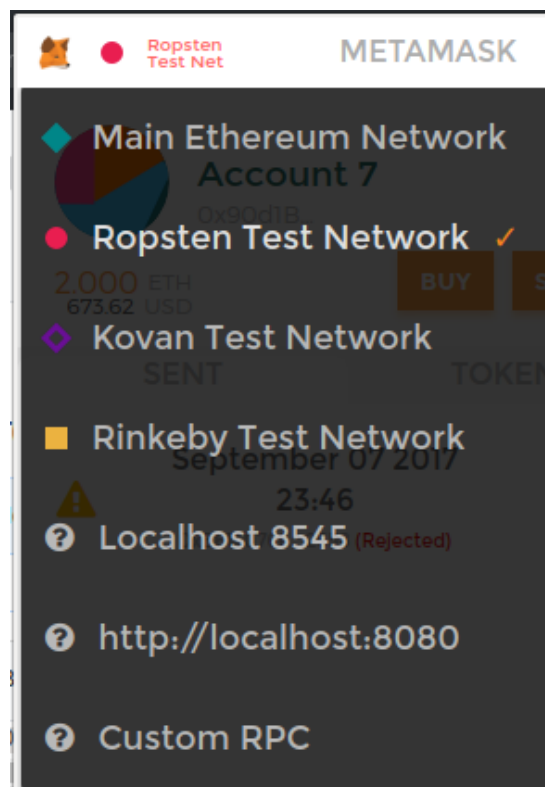
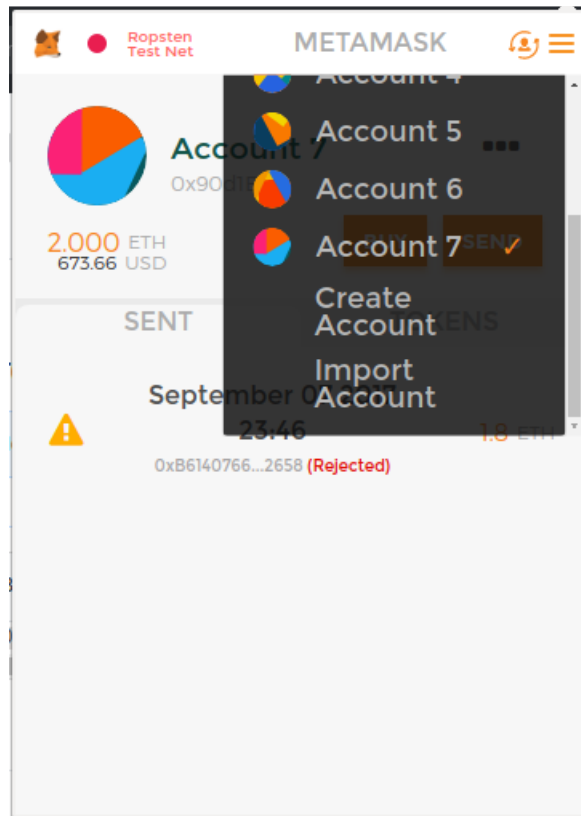



Figure 12, Users can change the network, choosing their own local node or a pre-supplied node from MetaMask





*Figure 13, User can change account, create new account or import an existing account with a private key or key file*



Ropsten  
Test Net


METAMASK




←

CONFIRM TRANSACTION

**Account 7**  
90d1B8...4745  
2.000 ETH  
673.66 USD



>



B61407...2658

Amount

1.900 ETH  
639.98 USD

Gas Limit

UNITS

Gas Price

GWEI

Max Transaction Fee

0.000630 ETH  
0.21 USD

Max Total

1.900 ETH  
640.19 USD

Data included: 0 bytes

RESET

SUBMIT

REJECT

Figure 14, Window that pops up every time a transaction needs signing from the user, giving them a chance to review the transaction before confirming and sending it.

## 5.3 Examples of Xcoin UI

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/tokenFactory'. The page title is 'Token Factory'. On the right side, there is a green notification box containing the text: 'Account: 0xe244d65e765ca61d6d4a9cctca34b8a24c0125f8' and 'Connected to private network. ID: 1504852254756'. The main content area contains several input fields for creating a new token:

- Name:** Input field with placeholder 'e.g. myCoin'.
- Initial amount:** Input field with placeholder 'e.g. 10000'.
- Consensus rule:** Input field with placeholder 'e.g. 75' and a '%' symbol.
- Issuance rate:** Input field with placeholder 'e.g. -2,100'.
- Initial Ether Balance:** Input field with placeholder 'e.g. 5 (ether)'.
- Upper Cap:** Input field with placeholder 'e.g. 100000'.

Figure 15, The user can create a new token, with the upper right green window confirming their account address and the network they are connected to

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/tokenSpace'. The page title is 'TokenSpace'. The main content area contains a dropdown menu with the text 'Please choose the TokenSpace you want to enter'. The dropdown list shows four options:

- theoCoin, 0x00d561ec2de548534c#72bddf35e3babe396198
- theoCoin, 0x00d561ec2de548534c#72bddf35e3babe396198
- greenCoin, 0xac2620f961c2193358ec293080f7d61464c29d2
- friendCoin, 0x799c2ceb82449456e02196652eeb1444f6d4679a

Below the dropdown menu, there are buttons for 'Login' and 'Sign Up'. At the bottom of the page, there is a section titled 'Or Go Back Home' with a button 'Hide/Show list of descriptions'. Below this button is a table with two columns: 'Token' and 'Description'.

Token	Description
theoCoin	This coin is for everyone with the name Theo
greenCoin	Let's save the environment yay!
friendCoin	who else wants to keep track of what they friends owe you?

Figure 16, The user can enter of the forums, or tokenspaces, by selecting the appropriate token from a drop-down list, with descriptions of the token given below



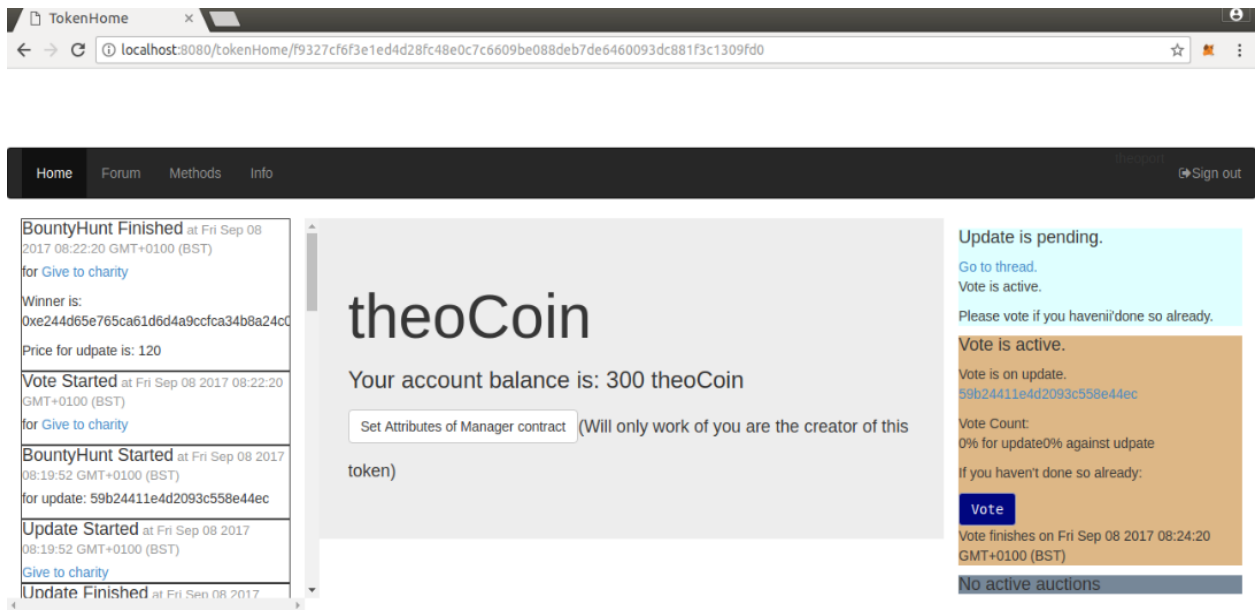


Figure 17, the tokenhome: in the middle is an indication of the user's account balance, while on the left is a history of events that have happened in the past, and on the right is a summary of which events are happening right now and what input is required of the user. Throughout Xcoin, blue buttons with white writing indicate blockchain interaction.

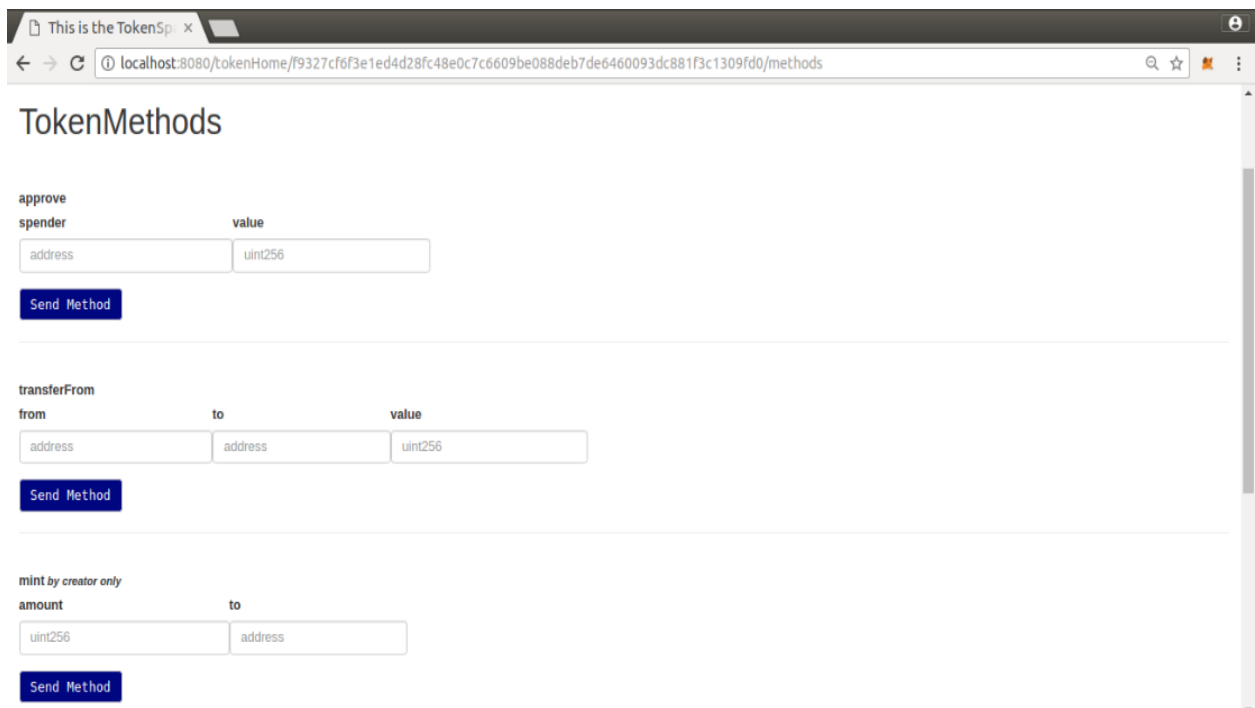


Figure 18, Inside the tokenspace, user has access to all public methods in token contract, with indication as to whether they are only accessible to the creator of the token. This page automatically updates itself when a token is updated and new methods are added, or existing methods modified

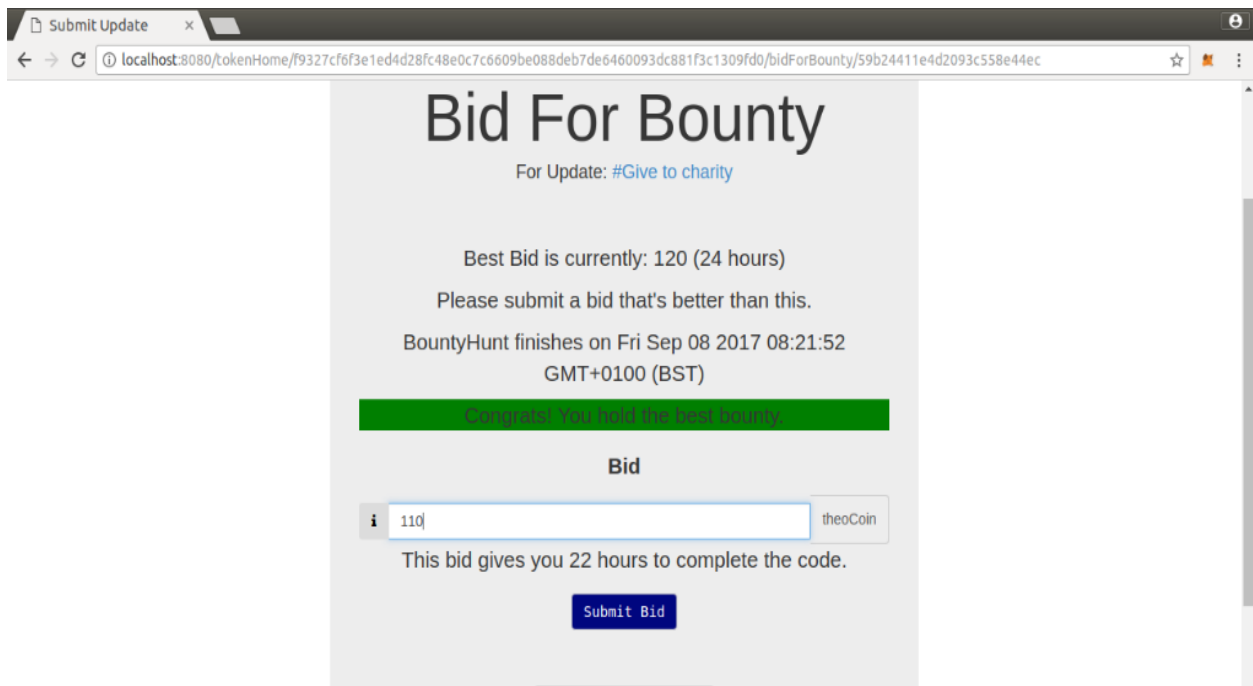


Figure 19, lets developers bid for bounty, indicates to the user whether they hold the current best bid and when the bountyHunt finishes. Also informs the user how time they are given for a given proposed price

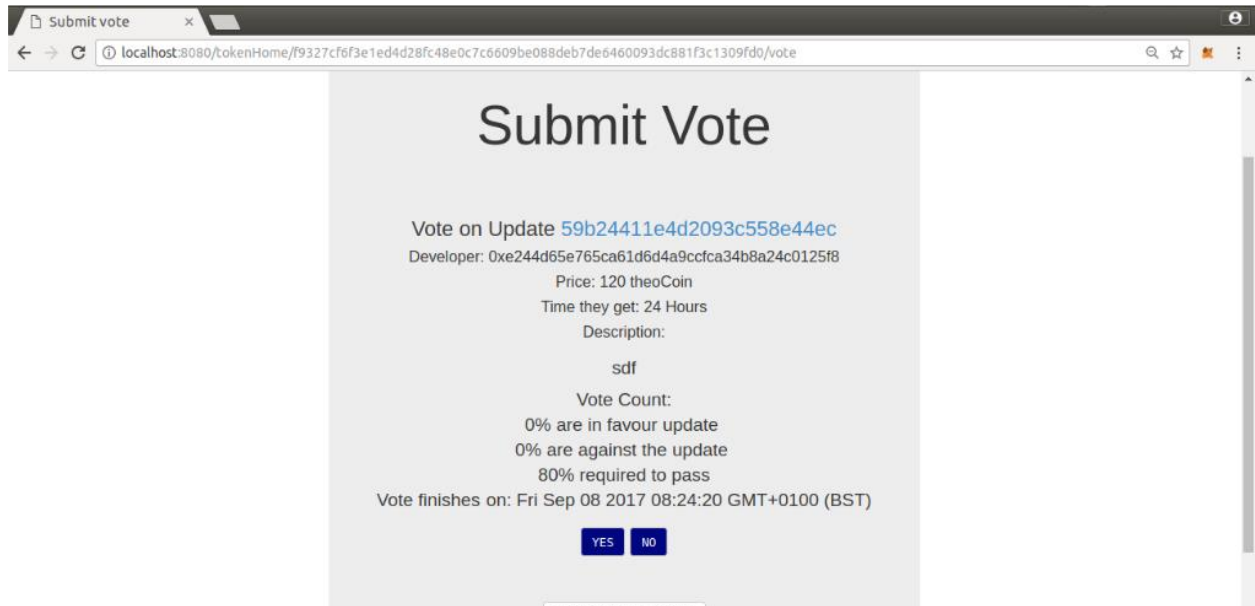


Figure 20, lets users submit votes. Gives a summary of what the vote is about, what the current voting stats are and provides a link to the thread for discussion.

## 6: Evaluation

### 6.1 Review of objectives

The objectives as outlined in the introduction are:

1. Introduce ideas of monetary policy and central banks

Objective was fulfilled in background section

2. Analyse concepts of blockchain-based networks with a focus on ethereum.

Objective was fulfilled in background section

3. Describe the idea of a decentralized autonomous organisation, drawing parallels between Xcoin and theDAO, and how symptomatic issues in DAOs will be avoided

Objective was fulfilled in background section

4. Design a prototype of the application with focus on functionality, user friendliness and decentralized structuring

The application was designed and all functions described in the summary of the design chapter implemented, with the exception of token mining, as this was realised to be unnecessary later in the project:

A. User can create new TokenManager and Token with customizable features

B. User can choose already existing Token and enter the corresponding forum with a username and password to prevent spamming

C. In Forum:

- a. Community is informed about latest events with clear instructions on how to participate
- b. Community can create a new topic for an update idea they have
- c. Community can start a bountyHunt for any existing update topic
- d. Community can receive an overview of the Token attributes
- e. Community can use all public methods of the Token contract including newly added features from updated Token
- f. Developer can bid for bounty
- g. Developer can submit update implementation and get paid for it
- h. Community can find bugs in update implementation submitted by developer
- i. Community can vote on legitimacy of found bugs
- j. Community can bid in token auction
- k. Community can choose to pay for transactions in token instead of ether

D. Smart Contracts can:

- a. *Token:*
  - i. Facilitate mining of token.
  - ii. Implement methods of ERC20 standard

- b. *TokenManager*:
  - i. Maintain the logic behind all activities that community and developer take part in
  - ii. Send regular information to front-end about current state
  - iii. Process outcome of activities mentioned in 3. and perform required response
  - iv. Facilitate transition between old contract Token to updated Token
- c. *TokenCreator*:
  - i. Create new TokenManagers and Tokens

E. Maintain a hybrid data storage system, combining benefits of server databases and blockchains

5. Deploy application to a private network that simulates the behaviour of an Ethereum network.

All functionality except auctions and the refund feature were tested on a private network for all possible use cases and performed as expected. Truffle in combination with TestRPC provided a quick and efficient deployment and testing facility, allowing the project to progress fast.

6. Deploy application to the Ethereum testnet and make site publicly available

This Objective was not achieved as the code was not ready for deployment to a public server. The contracts were deployed to the Ropsten network and lightly tested.

7. Review objectives, list potential problems and examine its feasibility, comparing it to similar projects

## 6.2 Potential problems

1. A developer can offer to implement an update for the lowest possible price, one token, guaranteeing her the job and then just not do it, making the community have to wait until the time window passes and they can finish the job. This could halt the development of a community.
  - Implemented solution: Fix the hourly wage of the developer, which means that making a bid of 1 token will give them an extremely short time window to implement the update and the community can quickly move on. A lazy developer could still just ignore the job, but the longer the window, the higher the price and the higher the incentive to actually implement the update.
2. A user can start a bountyHunt for an update that no one is interested in and therefore make the community wait until the voting time window passes.

- Implemented solution: Finish votes prematurely when required consensus or counter-consensus is reached. An interesting update can therefore be cancelled quickly by voting against it and the community can quickly move on. Proposer of update should also have to pay a small deposit and be rewarded for a successful vote to encourage detailed and well-formed descriptions of the update.

### 3. Voter Apathy

- No implemented solution. As described in the background section, communities that connect through ideals rather than monetary incentives might have a larger drive to cooperate and commit to the token. To combine monetary incentives with idealistic incentives, users could be required to hold a certain amount of token to be able to engage with the community, making them direct stakeholders of their involvement.

### 4. Members sell token and leave to create new community as soon the community votes against them, depriving community of its members as there will always be disagreements

- No implemented solution. In a world where people reorganise around currencies, token-membership could be subject to rules like: One is only allowed to be part of one community and can only change every x amount of months. For now, members have to learn that disagreements can bring a community forward if engaged with in a healthy discussion. Unfortunately, the internet has not shown this to be one of humans' strengths.

### 5. Finding bugs can be hard and the possibility of bug-free code and therefore no rewards means that no-one will invest the effort to actually try to find bugs

- No implemented solution. Rewards for finding bugs will have to be high enough to counter this issue.

### 6. Community members are not knowledgeable enough to vote on the legitimacy of a bug

- No implemented solution. Users have forums to discuss and the finder of the bug could try to explain it to other users. Developer admitting that it is indeed a bug could reassure users. Employing qualified experts to review code would allow developer and expert to cooperate and implement malicious code.

### 7. Majority robs the minority by passing an update that transfers funds of minority to accounts of majority

- No implemented solution. theDAO's solution to this problem was to allow users to perform a split and basically create a new DAO before the malicious update proposal is passed in the original DAO. A split has been considered but is deemed inappropriate as it would encourage users to

split away from the community as soon as a vote reaches a different outcome than they desired.

## 6.3 Gas costs

*Table 1, Prices of Xcoin functions*

Function	Gas price	Price in U.S. dollars, using an average gas price of 0.6 gwei*, as of September 8 <sup>th</sup> 2017
Setting a parameter	27978	\$0.00557
Transferring token	36277	\$0.00723
Starting a bounty	57857	\$0.0115
Ending a failed bounty	16875	\$0.00336
Bidding for bounty	50259	\$0.0100
Ending successful bounty	139656	\$0.0278
Voting on update	106235	\$0.0212
Voting on update causing it to succeed	72874	\$0.0145
Submitting an update	111202	\$0.022
Finding a bug	199995	\$0.0398
Voting on bug	91235	\$0.0182
Voting and causing bug to succeed	149240	\$0.0297
Finalising update	277321	\$0.0552
Transferring funds in change over	44684	\$0.00889
Killing old contract	26268	\$0.00523
Voting and causing update to fail	72676	\$0.0144
Voting and causing bug vote to fail	65249	\$0.0130
Deploying a TokenManager and Token to network	4322943	\$0.860

\*Recommended for a processing time of <20m [23]

\*\*Price for ether is \$332.24 [24]

## 6.4 Future work

### 6.4.1 Improved UI

At the moment, the UI is very basic and relies heavily on the Bootstrap framework, an aesthetically pleasing design that is often encountered online, but therefore not very unique. The UI of the prototype was primarily designed to showcase the functionality of the underlying processes and the next step is to give it a more inviting feel, making users feel more comfortable in using the software.

### 6.4.2 Optimisation of smart contracts

Up until this point, the main concern of the smart contracts was functionality, which has been achieved to a satisfactory level. The next step involves making a detailed analysis of the gas costs that are incurred by the smart contract operations. As was seen when measuring the gas usage, some functions seemed to spend more gas than others when intuitively that should not have been the case as they are less complex. An in-depth understanding of how Ethereum manages its different types of storage and what exact gas costs are associated with them will be helpful in optimizing the gas usage of the contracts before deploying them to the mainnet where gas costs money. Furthermore, the contracts should be passed to an expert for review of potential security threats and bugs.

### 6.4.3 Including a modifier *byVote()* in the Token and TokenManager contracts.

Instead of restricting the setting of contract parameters and the minting of token to the creator, a modifier could be designed that would start a vote on whether to implement that action or not. This could be implemented by passing the function identifier and arguments to the modifier, which it can then call if the vote is successful.

### 6.4.4 Allowing developers to update the forum itself

Currently, when a token is updated, its new methods are automatically added to the “Methods” tab in the application. This is done using the ABI, source code and address of the updated token. However, a user needs know what they are doing as they only have access to the type of input variables and name of the function. If the operation of a function requires further explanation, users will be unable to use this method. For this reason it would be ideal if the developer of the updated token is also asked to design a webpage that functions as UI for the new Token and its methods. This task could even be delegated to another developer once the contract is submitted.

### 6.4.5 Checking user input and cleaning the directory structure

Having been completely unfamiliar with web development, and having no examples on how to integrate a back-end directory structure into the directory that truffle creates for front-end applications, the directory structure ended up being slightly unorganised. This will cause

problems if other people wanted to continue working on the code or use it for their own projects, as it is planned to publish the code as an example of full stack Dapp development. Furthermore, it is currently assumed that users know what they are doing and don't supply the application with wrong types or formats. A thorough review of user input checking needs to be done.

#### **6.4.6 Leaving Ethereum behind**

Although implementing a prototype on the Ethereum blockchain has been a useful first step in the learning curve for Dapp development, the idea was originally to allow users to choose between implementing the token on top of an existing blockchain like Ethereum or Bitcoin, or to create an entirely new blockchain. The issue with using the Ethereum blockchain is that if a token gains value, it might start competing with ether, the underlying cryptocurrency of the Ethereum network, creating a conflict of interests for the Ethereum miner, who might not want to support the token, but is indirectly in charge of mining it. To avoid this dependency on a party that holds no stakes in the token, and on the contrary might be interested in the destruction of it, is dangerous.

Letting each community build their own blockchain is a possible solution. In this scenario, mining of token is required and updates to the token are implemented through a hard fork subject to a vote. If the vote succeeds, the hard fork is implemented, leaving behind the old token and anyone who refuses to follow the update.



## 7: Conclusion

The aim of this project was to create an online platform for users to create custom tokens and organise communities around them through forums. When making a new token, the creator is asked to supply a description that will help other users decide whether they are interested in joining that community. To avoid limiting the customisation of the token to variables that are deemed interesting by this project, communities can update the token after creation by reaching a consensus on a specific update idea and hiring a developer through a smart contract to implement the update for them. The update implementation cycle forms the core of the application and functioned as the main design goal.

To organise communities without the use of a central governing body, blockchain technology was used which allows code to be stored on a decentralized and immutable ledger that is maintained by a peer-to-peer network. That code, known as a smart contract, will rigorously follow the logic within it and reliably execute functions when given the right conditional inputs. This means that it can fulfil roles that usually require trust or close monitoring like counting votes, issuing currency or fulfilling contractual agreements.

In most of the world, citizens of a country are obliged to follow its fiscal and monetary policies, regardless of whether they agree with them or not. However, theoretically there should be no reason why we cannot create virtual economies that implement different sets of rules and which people can choose to join. Once joined, they will be directly affected by the decisions of their community, and such a platform could even function as experiment for different economic and monetary theories.

To implement the application, HTML, CSS and JavaScript were used for front-end development, while Node.js with a MongoDB database was used for back-end development. The truffle framework was used for the development and testing of smart contracts written in Solidity. The vast majority of developing was done using an Ethereum client called TestRPC that provides a private blockchain network, where a block is mined with every new transaction. Due to no time forced gaps in between blocks, it forms the perfect tool for development, but also exhibits different behaviour, not fully preparing the application for deployment to the Ethereum mainnet. The contracts were successfully deployed to the ropsten testnet towards the end of the project, and seem to function correctly.

The project managed to fulfil most of the design specifications, but was not made public on a server due to time constraints and its low importance given that the code is not ready to be publicized yet anyway. However, the plan is to continue working on it in the future and one day publicize it.

## References:

- [1] Andreas Antonopoulos, Bitcoin Expo 2014 – Keynote; Toronto, Ontario, Canada; April 2014
- [2] Dan Larimer, *Is the the DAO going to be DAO*, <https://steemit.com/crypto-news/@dan/is-the-dao-going-to-be-doa>
- [3] Tobin Hanson, *Futarchy: Vote Values, But Bet Beliefs*, <http://mason.gmu.edu/~rhanson/futarchy.html>
- [4] Eric Hughes, *S Cyberphunk's Manifesto*, <https://www.activism.net/cypherpunk/manifesto.html>
- [5] *Price of DogeCoin*, <https://coinmarketcap.com/currencies/dogecoin/>
- [6] Eximchain, *Quadratic Voting + Smart Contracts = Powerful governance model*, <https://medium.com/@eximchain/quadratic-voting-smart-contracts-powerful-governance-model-b8efa4ddeef1>
- [7] David Siegel, *E-Voting Resources*, <https://medium.com/@pullnews/e-voting-resources-510f6fbc6fa5>
- [8] John Koetsier, *ICO bubble? Startups are raising hundreds of millions of Dollars via Initial Coin Offerings*, <https://www.inc.com/john-koetsier/ico-bubble-startups-are-raising-hundreds-of-millio.html>
- [9] U.S. Securities and Exchange Commission, *Investor Bulletin: Initial Coin Offerings*, [https://www.sec.gov/oiea/investor-alerts-and-bulletins/ib\\_coinofferings](https://www.sec.gov/oiea/investor-alerts-and-bulletins/ib_coinofferings)
- [10] U.S. Securities and Exchange Commission, *Report of Investigation Pursuant to Section 21(a) of the Securities Exchange Act of 1934: The DAO*, <https://www.sec.gov/litigation/investreport/34-81207.pdf>
- [11] Canadian Securities Administrators, *CSA Staff notice 46-307: Cryptocurrencies*, [http://www.osc.gov.on.ca/documents/en/Securities-Category4/csa\\_20170824\\_cryptocurrency-offerings.pdf](http://www.osc.gov.on.ca/documents/en/Securities-Category4/csa_20170824_cryptocurrency-offerings.pdf)
- [12] Jeffrey E. Alberts & Bertrand Fry, *Is Bitcoin a Security?* [https://www.bu.edu/jostl/files/2016/01/21.1\\_Alberts\\_Final\\_web.pdf](https://www.bu.edu/jostl/files/2016/01/21.1_Alberts_Final_web.pdf)
- [13] *Statistics of Bountysource*, <https://www.bountysource.com/stats>
- [14] Michael Del Castillo, *Blockchain hiring difficulties becoming an industry concern* <https://www.coindesk.com/blockchain-hiring-difficulties-becoming-industry-concern/>
- [15] *Ethereum for Web Developers*, <https://medium.com/@mvmurthy/ethereum-for-web-developers-890be23d1d0c>
- [16] *ERC 20 Token Standard* [https://theethereum.wiki/w/index.php/ERC20\\_Token\\_Standard](https://theethereum.wiki/w/index.php/ERC20_Token_Standard)
- [17] ConsenSys, *Ethereum Contract Security Techniques and Tips*, <https://github.com/ConsenSys/smart-contract-best-practices>

- [18] *TestRPC: Fast Ethereum RPC client for testing and development*  
<https://github.com/ethereumjs/testrpc>
- [19] *Web3 JavaScript app API* <https://github.com/ethereum/wiki/wiki/JavaScript-API>
- [20] *Ethereum Alarm Clock*, <http://www.ethereum-alarm-clock.com/>
- [21] *Bitcoin Improvement Proposal*, <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>
- [22] *MetaMask Compatibility Guide*,  
<https://github.com/MetaMask/faq/blob/master/DEVELOPERS.md>
- [23] *ETH Gas Station*, <http://ethgasstation.info/>
- [24] *Ethereum (ETH) Price*, <https://www.coindesk.com/ethereum-price/>

## Appendix

### A.

The update implementation cycle :

0. No update pending
1. Start BountyHunt **by public**
  - a. Set update parameters in smart contract: ID and safetyHash
  - b. Set BountyHunt end time
2. End BountyHunt **by public**
  - a. If developer was found
    - i. Set update parameters in smart contract: developer, price and time window
    - ii. Go to 3
  - b. If developer was not found
    - i. Go to 0
3. Start vote **by TokenManager**
  - a. Freeze funds of *TokenManager*
  - b. Set vote end time
4. End Vote **by public**
  - a. If community votes yes
    - i. Go to 5
  - b. If community votes no
    - i. Go to 0
5. Start update **by TokenManager**
  - a. Set update parameter in smart contract: end time
6. Wait for developer

- a. If developer submits update before deadline
    - i. Go to 7
  - b. If developer misses deadline
    - i. Go to 0
- 7. Start BugHunt **by TokenManager**
  - a. Set update parameters in smart contract: address of new contract
  - b. Set BugHunt deadline
- 8. Wait for bughunters
  - a. If bug was found
    - i. Go to 11
  - b. If no bug was found
    - i. Go to 9
- 9. Finalise update **by developer**
  - a. Developer is paid
  - b. New contract is initialised
  - c. Give time for users to transfer funds
- 10. Kill old contract **by public**
  - a. Go to 0
- 11. Start vote **by TokenManager**
- 12. End vote **by TokenManager**
  - a. If community votes yes
    - i. Bug finder paid
    - ii. If developer exhausted attempts
      - 1. Go to 0
    - iii. If developer still has tries
      - 1. Add extension to deadline od update
      - 2. Go to 6
  - b. If community votes no
    - i. Go to 8