Developing a Blockchain eVoting Application using Ethereum

Dimitris Vagiakakos (E18019)
Department Of Digital Systems
University Of Piraeus
Piraeus, Attiki, Greece
dimitrislinuxos@gmail.com

Kostantinos Karahalis (E18065)
Department Of Digital Systems
University Of Piraeus
Piraeus, Attiki, Greece
ko.karahalis@gmail.com

Stavros Gkinos (E18043)
Department Of Digital Systems
University Of Piraeus
Piraeus, Attiki, Greece
stgkinos@gmail.com

ABSTRACT

Blockchain is on the receiving end of extensive attention in the last few years. This new technology serves as an established ledger which enables transactions that take place in a decentralized manner. Many applications that are linked to Blockchain keep on springing up, covering various fields including financial, political services, reputation system and Internet of Things (IoT) and so on.

1 INTRODUCTION

Blockchain is considered by many to be disruptive core technology and thus it has many uses. Even though, there are lots of researchers that have come to realize the great potential and importance of Blockchain, its groundwork is still progressive. In consequence, this paper reviews our extensive research on Blockchain and Ethereum, especially in the subject area of economics and politics(voting). Inspired by the way a voting system works, by business commerce and by the exploitable vulnerabilities in the security or the political corruption, as far as centralized systems are concerned, we have created a smart contract voting system(Ethereum-based) and we explore how the new technologies and various applications, that are associated with their functions, work.

2 BLOCKCHAIN

Blockchain is a decentralized peer-to-peer based network consisted of multiple blocks, which hold batches of valid transactions that are hashed and encoded into a Merkle tree. Each one of them includes the cryptographic hash value (usually 256-bit) of the previous block in the blockchain resulting in the creation of a link between them and in that order, the linked blocks, form a chain. This iterative process validates the integrity of the previous block, all the way to the first block, the genesis block. That block is the initial one in every blockchain-based system. It is the basis on which additional blocks are added to form a chain of blocks, hence the term blockchain. This block is sometimes referred to Block 0. Every block in a blockchain stores a reference to the previous block. In the case of Genesis Block, there is no previous block for reference. One distinctive feature of this block is that its hash value is added to all new transactions in a new block. As for the current block, it contains the hash value of the previous one, it's own hash value and the user's transaction data. Peers supporting the database have different versions of the history from time to time. They keep only the highest-scoring version of the database known to them. Whenever a peer receives a higher-scoring version they extend or overwrite their own database and re-transmit the improvement to their peers. There is never an absolute guarantee that any particular entry will remain in the best version of the history forever. Blockchains are typically built to add the score of new blocks onto old blocks and are given incentives to extend with new blocks rather than overwrite old blocks. Therefore, the probability of an entry becoming superseded decreases exponentially as more blocks are built on top of it, eventually becoming very low.

2.1 Nodes

In a Blockchain many people are distinguished according to their respective names and roles. The authority of a person usually refers to the possession and continuous updating of part or all of the chain files and a second responsibility refers to the right to apply for new blocks in the chain. Nodes which protect part of the chain are simply called nodes, while nodes that protect the entire chain are called master nodes. Both of these, use their storage space locally. Those node managers who perform the second responsibility are called miners and this process is called mining. All nodes are responsible for checking the legitimacy of the blocks being routed to them and forwarding them to neighbour nodes. Network users either they are these nodes or not, they create and channel useful data into it.

2.2 Blockchain Applications

As it was mentioned earlier, this technology has various real-life uses and some distinctive features. Many of those who have used Blockchain know its primary use, the cryptocurrency. However, it is so much more than the foundation for it, since it has provided us with the convenience to secure sharing of medical data, cross-border payments, personal identity security, anti-money laundering tracking system, voting mechanism, advertising insights, supply chain and logistics monitoring etc.

2.3 Decentralization

In a blockchain, each node has a full record of the data that has been stored on the blockchain since its inception. By using the method of storing data throughout its peer-to-peer network, the blockchain minimizes the risks that come with data being held in a central way by a lot. This peer-to-peer network that blockchain utilizes has so few vulnerabilities on the grounds that every time a successful attempt to modify the data of a block is taking place, the blockchain network is going to detect it and the validation process of the current block and the next blocks' hash value will be disapproved and therefore all nodes reject the chain, making it almost impossible for it to be corrupted. This is due to the fact that, Blockchain security methods include the use of public-key cryptography, a public key (a long, random-looking string of numbers) is an address on the blockchain. Value tokens sent across the network are recorded as belongings to that address. A private key is like a secret word-key that gives its owner access to their digital assets or the means to interact with the various capabilities that blockchain offers. Data stored on the blockchain is incorruptible. Every node in a decentralized system has a copy of the blockchain. Data quality is maintained by massive database replication and computational trust, as a centralized copy does not exist.

2.4 Proof Of Work

Proof-of-work is the algorithm that secures many cryptocurrencies, including Bitcoin. For the purpose of adding a new block, which bother transactions waiting for validation, into the blockchain miners need to solve a "puzzle" using computational power. A new block is accepted by the network each time a miner comes up with a new winning proof-of-work which is so difficult that miners use expensive and specialized computers. These computers require and utilize a large amount of electrical power, that provokes the environment, which means that countries, that offer inexpensive electric power have the ability to mine cryptocurrencies, using PoW, in much lower cost. As a consequence these states dominate each PoW networks ending up in a more centralised network. Proof of Work blockchains provide adequate security only if there is a large network of miners competing for block rewards. In case of a conflict, we always have to trust the longest chain of blocks because this means that it has the greatest computational work of all conflicted blockchain. In case the network in a large area, that hosts a considerable number of miners (also known as Mining Pools), breaks down the PoW network is losing a significant quantity of miners that make this system safe, as a result the possibility that a hacker could gain a simple majority of the network's computational power and stage grows up, what is known as a 51% attack.

2.5 Proof Of Stake

Proof of Stake (PoS) is a type of consensus mechanisms by which a cryptocurrency blockchain network achieves distributed consensus. In PoS-based cryptocurrencies the creator of the next block is chosen via various combinations of random selection or wealth or age. While the overall process remain the same as proof of work (POW), the method of reaching the "target" is different. In Power of Stake, instead of miners from Proof of work, there are validators. The blockchain keeps track of a set of validators, and anyone who holds the blockchain's base cryptocurrency (in Ethereum's case, ether) can become a validator by sending a special type of transaction that locks up their ether into a deposit. The validators take turns proposing and voting on the next valid block, and the weight of each validator's vote depends on the size of its deposit (i.e., stake). Importantly, a validator risks losing their deposit if the block they staked it on is rejected by the majority of validators. Conversely, validators earn a small reward, proportional to their deposited stake, for every block that is accepted by the majority. The validators lock up some of their Ether as a stake in the ecosystem. The validators bet on the blocks that they will be added next to the chain. When the block gets added, the validators get a block reward in proportion to their stake. The main advantages of the Proof of Stake algorithm are energy efficiency and security. A greater number of users are encouraged to run nodes since it is easy and affordable. This along with the randomization process also makes the network more decentralized, since mining pools are no longer needed to mine the blocks. Thus, PoS forces validators to act honestly and follow the consensus rules, by a system of reward and punishment. The major difference between PoS and PoW is that the punishment in PoS is intrinsic to the blockchain (e.g., loss of staked ether), whereas in PoW the punishment is extrinsic (e.g., loss of funds spent on electricity).

3 CRYPTOCURRENCY

In the last century, what we see before us is a world of an everlasting change in many aspects of life and of course in the technological field. One great example of the progress in the technological field is the invention and advance of the digital coin (ether, bitcoin, ...,etc.). However, a guaranteed method of a solid way of exchange is called for and that is cryptocurrency. Cryptocurrency is a digital asset, which is secured by cryptography and that makes it nearly impossible to counterfeit or double-spend. It is designed to work as a medium of exchange wherein individual coin ownership records are stored in a ledger existing in a form of computerized database using strong cryptography to secure transaction records, to control the creation of additional coins, and to verify the transfer of coin ownership. The defining feature of cryptocurrencies is that they are comprised of decentralized banking systems, which means that they are not controlled by any central authority, rendering them theoretically immune to government interference or manipulation. As a result, when executed with decentralized control each cryptocurrency works through distributed ledger technology, typically a blockchain, that serves as a public financial transaction database.

3.1 Basic Architecture of Cryptocurrency

The earlier mentioned decentralized cryptocurrency is manufactured by the cryptocurrency system as a whole, at a rate which is defined when the system is created and is publicly known. In centralized banking and economic systems, corporate boards or governments control the supply of currency by printing units of fiat money or demanding additions to digital banking ledgers. In the case of decentralized cryptocurrency, companies or governments cannot produce new units, and have not so far provided backing for other firms, banks or corporate entities which hold asset value measured in it. Most cryptocurrencies are designed to gradually decrease production of that currency, placing a cap on the total amount of that currency that will ever be in circulation. Compared with ordinary currencies held by financial institutions or kept as cash on hand, cryptocurrencies can be more difficult for seizure by law enforcement.

3.2 Wallets

Also, something which should be mentioned is the use of wallets in cryptocurrency. A cryptocurrency wallet stores the public and private "keys" and "addresses" which can be used to receive or spend the cryptocurrency. With the private key, it is possible to write in the public ledger, effectively spending the associated cryptocurrency. With the public key, it is possible for others to send currency to the wallet. Another group of key players are the so-called “wallet providers”. Wallet providers are those entities that provide cryptocurrency users digital wallets or e-wallets which are used for holding, storing and transferring coins. Simply put, a wallet holds a cryptocurrency user’s cryptographic keys. A wallet provider typically translates a cryptocurrency user’s transaction history into an easily readable format, which looks much like a regular bank account. In reality, there are several types of wallet providers:
i) Hardware wallet providers that provide cryptocurrency users with specific hardware solutions to privately store their cryptographic keys (e.g. Ledger Wallet108, . . .)
ii) Software wallet providers that provide cryptocurrency users with software applications which allow them to access the network, send and receive coins and locally save their cryptographic keys (e.g. Jaxx109).

4 BITCOIN

Bitcoin is the first cryptocurrency that is underpinned by a kind of distributed ledger known as blockchain. This ledger consists of a record of all bitcoin transactions, arranged in sequential blocks, so that no user is allowed to consume any of their holdings twice. In order to avert tampering, the ledger is public, an altered version would be rejected by other users. It is a decentralized digital currency without a central bank or single administrator that can be sent from user to user on the peer-to-peer bitcoin network without the need for intermediaries. Transactions are verified by network nodes through cryptography and recorded in a public distributed ledger called a blockchain. Bitcoins are created as a reward for a process known as mining. They can be exchanged for other currencies, products, and services. Generating just any hash for a set of bitcoin transactions would be negligible for a 21st century computer, so the bitcoin network sets a certain level of adversity everytime a block has been "mined". More specifically a new block is mined approximately every 10 minutes. This block is added to the blockchain by generating a valid hash. The level of adversity is accomplished by establishing a "target" for the hash: The lower the target, the smaller the set of valid hashes and the harder it is to generate-"mine" one.

4.1 Mining Bitcoin

Mining is an antagonistic process but is more like a lottery than a race. Generating just any hash for a set of bitcoin transactions would be negligible for a 21st century computer and so the bitcoin network sets a certain level of adversity everytime a block has been "mined". More specifically a new block is mined approximately every 10 minutes. This block is added to the blockchain by generating a valid hash. The level of adversity is accomplished by establishing a "target" for the hash: The lower the target, the smaller the set of valid hashes and the harder it is to generate-"mine" one.How do miners be certain that they generated a hash below the target? They adjust the input by adding an integer, called nonce (number used once). When a valid hash is found, it is broadcast to the network and the block is added to the blockchain. Someone will generate an acceptable proof of work around every ten minutes, but we can not ever know who that someone will be. Miners pool together to raise their chances of mining blocks which generates transaction fees and a reward of newly created bitcoins for a limited period of time.

5 ETHEREUM

Ethereum is an open source blockchain, globally decentralized computing infrastructure that executes programs, called smart contracts. Smart contracts are deployed in a runtime environment, called Ethereum Virtual Machine. Ether (Ethereum’s Network coin) is the second largest cryptocurrency by market capitalization, after Bitcoin. As with the other cryptocurrencies, the validity of each Ether is provided by a blockchain, which is an incessantly growing list of records, called blocks, which are linked and secured using hash functions, like Keccak-256 hash. Hash. A cryptocurrency wallet stores the private keys and the addresses. Addresses can be used to receive or spend Ethereum. Having the private key, the blockchain can be written, making an Ether transaction. Due to decentralized framework of the Ethereum developers are able to build decentralized applications with built-in economic functions, as it provides high availability, auditability, transparency and neutrality, while simultaneously reduces or eliminates counter intelligence and certain counter party risks.

5.1 Ether

As we have already mentioned, Ether is the cryptocurrency generated by the Ethereum protocol as a reward to miners for adding blocks to the blockchain and is the only currency accepted in the payment of transaction fees, which also go to miners. The block reward together with the transaction fees is the inducement of miners to keep the blockchain growing, and therefore ether is fundamental to the operation of the network. Each Ethereum account has an ETH balance and may send ETH to any other account.

5.2 Addresses

Ethereum addresses are consisting of, the prefix "0x", concatenated with the rightmost 20 bytes of the Keccak-256 hash of the ECDSA (Elliptic Curve Digital Signature Algorithm) public. ECDSA is a cryptographic algorithm is used by Ethereum in order to ensure that funds can only be spent by their owners. In hexadecimal, 2 digits represent a byte, meaning addresses contain 40 hexadecimal digits. Smart contract addresses are in the same format, however, they are determined by sender and creation transaction nonce. User accounts are indistinguishable from contract accounts, given only an address for each and no blockchain data. Any valid Keccak-256 hash put into the described format is valid, even if it does not correspond to an account with a private key or a contract.

5.3 Gas

Gas is a unit of account within the EVM used in the calculation of a transaction fee, which is the amount of ETH a transaction's sender must pay to the miner who includes the transaction in the blockchain. Practically, Gas it's a separate virtual currency with its own exchange rate against Ether. This computation model requires some form of metering in order to avoid denial-of-service (DDoS) attacks inadvertently resource-devouring transactions. The price of gas has relevance for the time for a transaction to be confirmed. The higher gas price, the faster the transaction is likely to be confirmed. As a matter of fact, gas price can be set to zero. Nothing in the protocol prohibits free transactions. But then, it is very likely to never been confirmed. During periods of low demand for space in a block, such transactions might very well get mined.

5.4 Transaction

Data committed to the Ethereum Blockchain signed by an originating account, targeting a specific address. The transaction contains metadata such as the gas limit for that transaction.

5.5 Smart Contracts

Smart contracts are computer programs with some extra unique features. The most significant feature of Smart Contracts is that they are immutable. Once deployed, the code of a smart contract cannot be modified or change. The only way to alter the code of a smart contract is to deploy a new instance. Moreover, the outcome after the execution of a smart contract can't be contradictory. The outcome of the execution is the same for everyone who runs it. The EVM runs as a local instance of every Ethereum node, however as all of the instances of the EVM operate on the same initial state and produce the same final state. That makes the Ethereum Network a Decentralized "world computer". Importantly, contracts only run if they are called by a transaction

5.6 Ethash

Ethash is the Proof Of Work function in Ethereum-based currencies. The Ethash algorithm relies on a pseudorandom dataset, initialized by the current blockchain length. This is called a DAG and is regenerated every 30,000 blocks. Use of consumer-level GPUs for carrying out the PoW on the Ethereum network means that more people around the world can participate in the mining process. The more independent miners there are the more decentralized the mining power is, which means we can avoid a situation like in Bitcoin, where much of the mining power is concentrated in the hands of a few large industrial mining operations. In fact, there is a deliberate handicap on Ethereum's proof of work called the difficulty bomb, intended to gradually make proof-of-work mining of Ethereum more and more difficult, thereby forcing the transition to Proof of stake.Ethereum's Proof Of Stake Algorithm called Casper.

5.7 Compared to Bitcoin

Ethereum has several differences from Bitcoin. First of all, Bitcoin is a singular form of digital money where users can send, receive, and hold only bitcoins. Ethereum is a smart contract platform which allows entities to leverage blockchain technology to create numerous different digital ledgers and can be used to create additional cryptocurrencies that run on top of its blockchain. As a result, Bitcoin is aimed to only be a type of cryptocurrency, compared to Ethereum where it can also run applications. As has already mentioned, Ethereum's coin is Ether and its block time is 13 seconds, compared to 10 minutes for Bitcoin. Instead of Power of Proof, Ethereum uses Ethash until it will be replaced by Proof of Stake. Furthermore, in Ethereum, transaction fees differ by computational complexity, bandwidth use and storage needs, while bitcoin transactions compete by means of transaction size in bytes. And finally, Ethereum uses an accounting system where values in Wei (the smallest denomination of 1 Ether, 1 ETH = 1018 Wei) are debited from accounts and credited to another, as opposed to Bitcoin's UTXO system, which is more analogous to spending cash and receiving change in return.

5.8 Solidity

Solidity is an object-oriented, high-level language, explicitly for writing smart contracts with features to directly support execution on various blockchain platforms and especially, in the decentralized environment of the Ethereum world computer. Solidity was influenced by C++, Python and JavaScript program languages and is designed to target the Ethereum Virtual Machine (EVM).The main "product" of the Solidity project is the Solidity compiler (solc) which converts programs written in the Solidity language to EVM bytecode. The language is still in constant flux and things may change in unexpected ways to resolve such issues. Solidity offers a compiler directive known as a version pragma that instructs the compiler that the program expects a specific compiler (and language) version. The Solidity compiler reads the version pragma and will produce an error if the compiler is incompatible with the current version pragma. Also, it should be mentioned that minor updates of Solidity (For example, from 0.8.0 to 0.8.1) are compatible with their main version (0.8.x.). Each Pragma directives are not compiled into EVM bytecode. Version pragma is only used by the compiler to check the compatibility. In order to add version pram, we type: Pragma solidity ˆ VERSION_NUMBER;

5.9 Web 3

Web3 represents a new vision and focus for web applications: from centrally owned and managed applications, to applications built on decentralized protocols. The main advantage of Web 3 is that it attempts to address the biggest problem that has resulted from Web 2: the collection of personal data by private networks which are then sold to advertisers. With Web 3, the network is decentralized, so no such entity controls it, and the Decentralized Applications (DApps) that are built on top of the network are open. The devoutness of the decentralized web means that no single party can control data or limit access. Anybody is able to build and connect with different Decentralized Applications without permission from a central company. Web 3 uses A Ethereum JavaScript API, called Web3.js which is a collection of libraries that allow users to interact with a local or remote Ethereum Node Using HTTP, IPC or WebSocket.

5.10 Ganache

Ethereum smart contracts are programs executed within the context of transactions on the Ethereum blockchain. Ganache allows users to recreate a blockchain environment is their local systems and test the smart contracts that they created. It simulates the features of a real Ethereum network, namely it generates several addresses and each one of them owns 100 ether with their own private key. Whenever a transaction is performed it gets added to block in order to show the current block number in that blockchain. Ganache gives you ,every time you start it, a certain amount of Ether and after every transaction it shows you what the gas price was for the transaction that occured.

5.11 Metamask

Metamask is a browser plugin that is used as an Ethereum software wallet that serves as the primary user interface to Ethereum Network. Once a user installs the Metamask in his browser he is able to store Ether and other ERC-20 tokens and make transactions to any Ethereum address. The wallet can also be used by the user to interact with Decentralized Applications. Metamask, just like other "web3" based applications, aims to decentralize control over personal data and increase user privacy. Metamask can be used to run both local and remote Ethereum Blockchains. This function with the help of Ganache, provide a way to do transcactions to our local Ethereum Network in order to test our Decentralized Applications we develop.

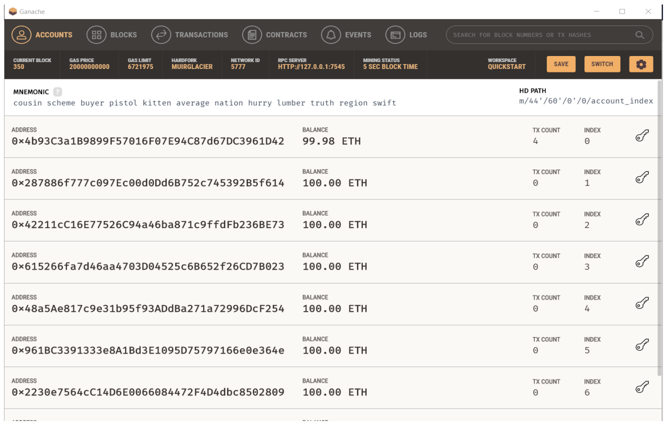
5.12 Remix - Ethereum IDE

Remix, one of the tools that helped us to test our smart contracts, is the official online IDE provided by Ethereum.org and it is used to create Ethereum smart contracts in the Solidity programming language. It has several built-in Solidity compiler versions and it runs in the web browser. Furthermore, it is used to provide a connection with Web3.js to interact with wallets like Metamask or with our local Blockchain network. For instance, we exploited its functionality to connect with our local Blockchain Ganache by entering its address. Hence, this enables it to deploy smart contracts or to call contract functions.

5.13 Truffle

Truffle is a world-class development environment, testing framework, and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM), aiming to make a developer's life easier. In other words, Truffle is a development environment, testing framework, and asset pipeline all rolled into one. It is based on Ethereum Blockchain and is designed to facilitate the smooth and seamless development of Decentralised Applications. With Truffle, you can compile and deploy Smart Contracts, inject them into web apps, and also develop front-end for Decentralised Applications. Today,Truffle is one of the most widely used IDEs for Ethereum Blockchain. The Truffle Suite comprises of three core elements:
1) Truffle: The actual development environment that integrates compilation, testing, and deployment of Smart Contracts.
2) Ganache: Ganache's details of how it works were mentioned earlier.
3) Drizzle: It is an assortment of front-end libraries that offer useful components for developing web applications that can seamlessly connect with the Smart Contracts. Truffle is quickly becoming the most useful framework for blockchain development. packs together the best modules and tools to streamline smart contract creation, compilation, testing, and deployment onto Ethereum. Furthermore, it supports front-end development using a Redux store that automatically syncs with the contract data. Therefore, integration with React is simple and easy. Written in JavaScript, Truffle modularizes key features to abstract away the complexity and cognitive load. The features that make Truffle Ethereum one of the most widely used IDEs for Ethereum Blockchain are:
1)It has built-in support for compiling, deploying and linking Smart Contracts,
2)It allows for automated contract testing with Mocha and Chai,
3)The Truffle Console allows you to work with your compiled contracts in a hassle-free manner
4)It has a configurable build pipeline that supports both console apps and web apps,
5)It comes with built-in support for JavaScript, CoffeeScript, SASS, ES6, and JSX,
6)It has generators that help in the creation of new contracts and tests (for instance rails generate),
7)It has a script runner that allows you to run JS/Coffee files, including your Smart Contracts,
8)It allows for the instant rebuilding of assets during the development stage,
9)It enables contract compilation and deployment using your preferred choice of RPC client and finally it supports both network and package management.

6 DECENTRALIZED EVOTING SMART CONTRACT ON ETHEREUM BLOCKCHAIN



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
350	20000000000	6721975	MUIRGLACIER	5777	HTTP://127.0.0.1:7545	5 SEC BLOCK TIME	QUICKSTART

MNEMONIC
cousin scheme buyer pistol kitten average nation hurry lumber truth region swift

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×4b93C3a1B9899F57016F07E94C87d67DC3961D42	99.98 ETH	4	0
0×287886f777c097Ec00d0Dd6B752c745392B5f614	100.00 ETH	0	1
0×42211cC16E77526C94a46ba871c9ffdFb236BE73	100.00 ETH	0	2
0×615266fa7d46aa4703D04525c6B652f26CD7B023	100.00 ETH	0	3
0×48a5Ae817c9e31b95f93ADdBa271a72996DcF254	100.00 ETH	0	4
0×961BC3391333e8A1Bd3E1095D75797166e0e364e	100.00 ETH	0	5
0×2230e7564cC14D6E0066084472F4D4dbc8502809	100.00 ETH	0	6

6.1 Preview of Source Code

We have created a smart contract with the name Election.sol, which handles the process of our election.Our eVoting Smart Contract is written in Solidity version 0.8.0 . At first, with the struct Candidate the details of each candidate are defined. Namely their id, name and the total number of their votes. Then via the use of mapping, in other words a type of reference to arrays and structs, we can load and fetch the number of our voters. Furthermore, the same process is used for our candidates-politicians. Then, we add new candidates into our structure Candidates through the function NewCandidate ("Name of candidate"). However, we have to ensure that the voters won't vote more than one time and in order to utilize this we used the function vote. This function takes their address and if it has scanned it more than once then the candidate's extra vote is considered invalid. Also, the function updates the total number of votes each time the voted event is triggered (our smart contract is being executed).
Now, as for the Migrations.sol contract it is essentially used to upgrade the addresses, namely through the function upgrade it adds and updates the new address(transaction) into our smart election voting application.

```
 //"SPDX-License-Identifier: UNLICENSED"
pragma solidity ^0.8.0;

contract Election {

    // Create the candidates structure

    struct Candidate {
        uint id;
        string name;
        uint CountOfVotes;
    }

    // Through the use of mapping we load and fetch
         ↪ the number of our voters
    mapping(address => bool) public voters;

    // As mentioned above, the same process is being
         ↪ done
    // for our future politcians
    mapping(uint => Candidate) public candidates;

    // Store Candidates Count
    uint public CountOfCandidates;

    // voted event
    event votedEvent (
        uint indexed _candidateId
    );

    constructor() public {
        NewCandidate("Candidate 1");
        NewCandidate("Candidate 2");
        NewCandidate("Candidate 3");
        NewCandidate("Candidate 4");
        NewCandidate("Candidate 5");
    }

    function NewCandidate (string memory _name)
         ↪ private {
        CountOfCandidates ++;
        candidates[CountOfCandidates] = Candidate(
             ↪ CountOfCandidates, _name, 0);
    }

    function vote (uint _candidateId) public {
        // require that they haven't voted before
        require(!voters[msg.sender]);

        // require a valid candidate
        require(_candidateId > 0 && _candidateId <=
             ↪ CountOfCandidates);

        // check if a voter has already voted
        voters[msg.sender] = true;

        // Update the total number of votes
        candidates[_candidateId].CountOfVotes ++;

        // trigger voted event
        emit votedEvent(_candidateId);
    }
}
```

6.2 Preparation of the Local Blockchain

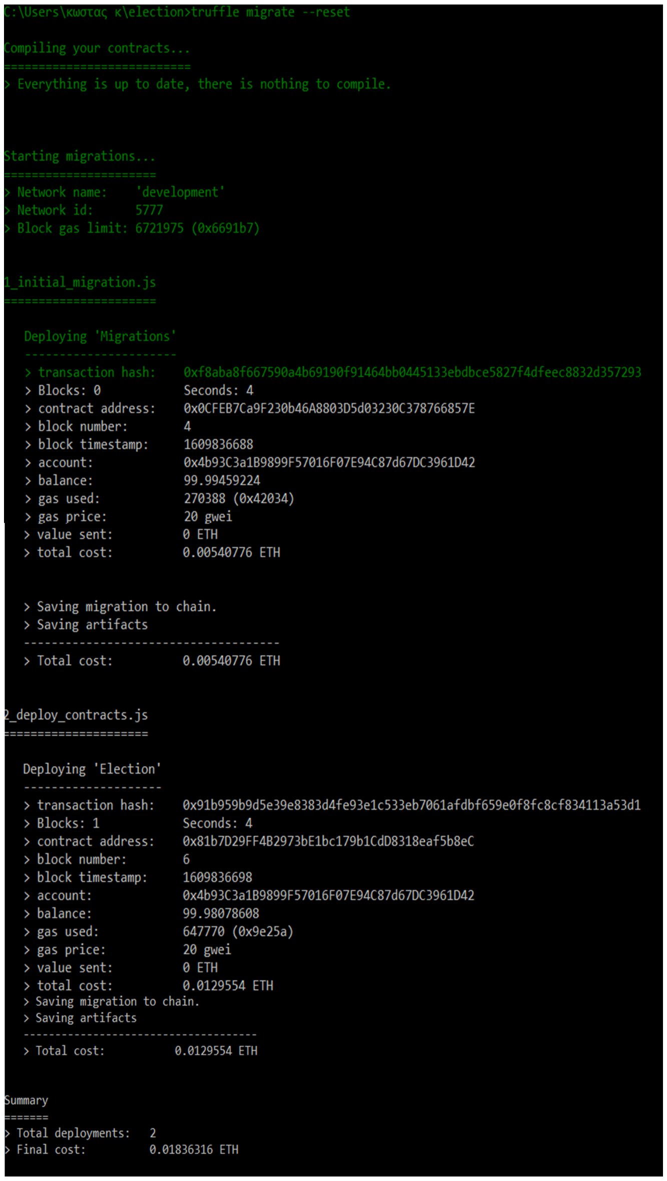
For the development purposes, our smart election contract deploys locally on Windows 10 in Node.js (all the commands are executed via the command line!!!). Firstly, we open the directory where the code is located and then we run Ganache, as seen from the image above. Ganache is used to create a local blockchain in our computer for our development purposes. Our local Blockchain with our localhost IP: HTTP://127.0.0.1 , running in port 7545 has its own local addresses and each one of them has an amount of 100ETH. It is of highly importance to have an amount of ETH on the grounds that without ether, we are unable to execute a transaction which is in our case a vote. Also, it should be mentioned that every voter cannot vote more than one time.

6.3 Compilation and Deployment with Truffle

Truffle is able to compile contracts written in Solidity and after everything is correct in syntax of our code, these contracts can be deployed in truffle with the command. Truffle migrations facilitate us to "push" our eVoting Decentralized Application (Dapp) to our local Ethereum Blockchain that we have already created:

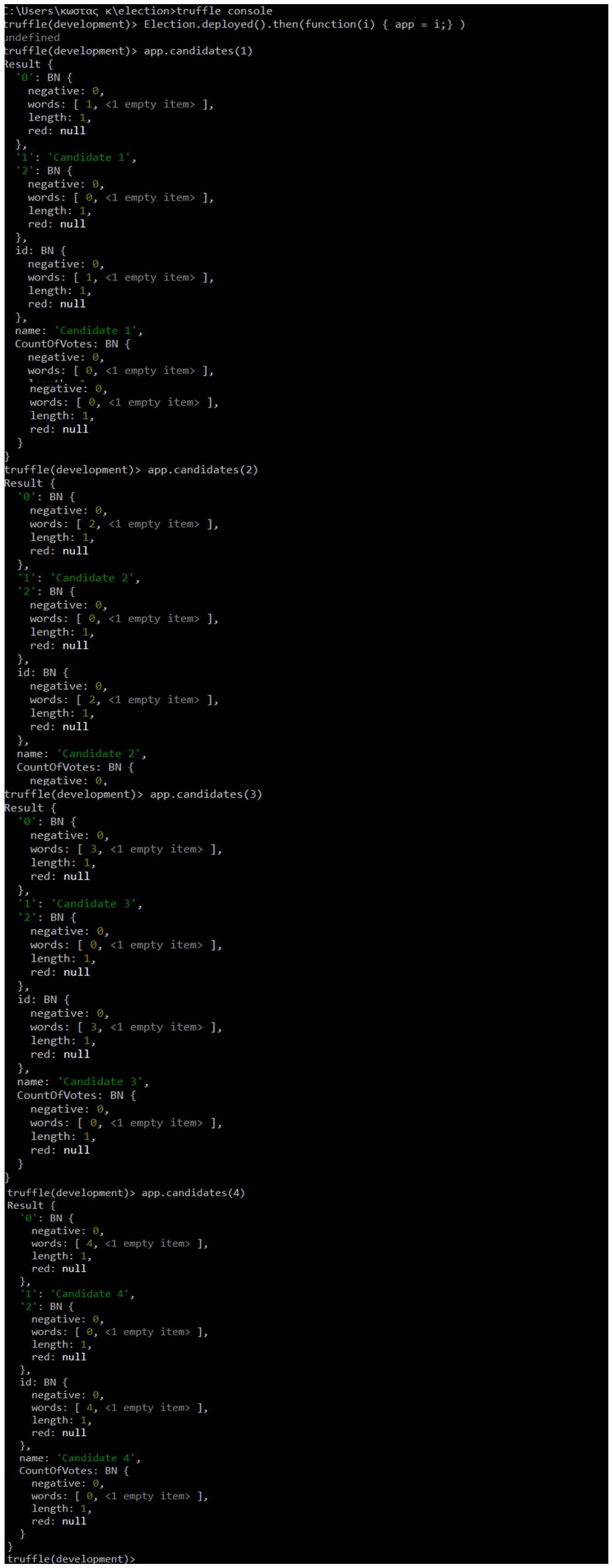
truffle migrate –reset

Truffle deploys the contracts Election.sol and Migrations.sol into local Blockchain (–reset: resets all smart contracts) and displays the details of the initial transaction, as it can be seen from the images below:



```
C:\Users\κωστας κ\election>truffle migrate --reset

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
======================

   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0xf8aba8f667590a4b69190f91464bb0445133ebdbce5827f4dfeec8832d357293
   > Blocks: 0            Seconds: 4
   > contract address:    0x0CFEB7Ca9F230b46A8803D5d03230C378766857E
   > block number:        4
   > block timestamp:     1609836688
   > account:             0x4b93C3a1B9899F57016F07E94C87d67DC3961D42
   > balance:             99.99459224
   > gas used:            270388 (0x42034)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00540776 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00540776 ETH


2_deploy_contracts.js
=====================

   Deploying 'Election'
   --------------------
   > transaction hash:    0x91b959b9d5e39e8383d4fe93e1c533eb7061afdbf659e0f8fc8cf834113a53d1
   > Blocks: 1            Seconds: 4
   > contract address:    0x81b7D29FF4B2973bE1bc179b1CdD8318eaf5b8eC
   > block number:        6
   > block timestamp:     1609836698
   > account:             0x4b93C3a1B9899F57016F07E94C87d67DC3961D42
   > balance:             99.98078608
   > gas used:            647770 (0x9e25a)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0129554 ETH
   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.0129554 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.01836316 ETH
```

Now, we are able to enter the console using the command truffle console so as to interact with the smart contract as seen downwards:



```
:\Users\κωστας κ\election>truffle console
truffle(development)> Election.deployed().then(function(i) { app = i;} )
undefined
truffle(development)> app.candidates(1)
Result {
  '0': BN {
    negative: 0,
    words: [ 1, <1 empty item> ],
    length: 1,
    red: null
  },
  '1': 'Candidate 1',
  '2': BN {
    negative: 0,
    words: [ 0, <1 empty item> ],
    length: 1,
    red: null
  },
  id: BN {
    negative: 0,
    words: [ 1, <1 empty item> ],
    length: 1,
    red: null
  },
  name: 'Candidate 1',
  CountOfVotes: BN {
    negative: 0,
    words: [ 0, <1 empty item> ],
    negative: 0,
    words: [ 0, <1 empty item> ],
    length: 1,
    red: null
  }
}
truffle(development)> app.candidates(2)
Result {
  '0': BN {
    negative: 0,
    words: [ 2, <1 empty item> ],
    length: 1,
    red: null
  },
  '1': 'Candidate 2',
  '2': BN {
    negative: 0,
    words: [ 0, <1 empty item> ],
    length: 1,
    red: null
  },
  id: BN {
    negative: 0,
    words: [ 2, <1 empty item> ],
    length: 1,
    red: null
  },
  name: 'Candidate 2',
  CountOfVotes: BN {
    negative: 0,
truffle(development)> app.candidates(3)
Result {
  '0': BN {
    negative: 0,
    words: [ 3, <1 empty item> ],
    length: 1,
    red: null
  },
  '1': 'Candidate 3',
  '2': BN {
    negative: 0,
    words: [ 0, <1 empty item> ],
    length: 1,
    red: null
  },
  id: BN {
    negative: 0,
    words: [ 3, <1 empty item> ],
    length: 1,
    red: null
  },
  name: 'Candidate 3',
  CountOfVotes: BN {
    negative: 0,
    words: [ 0, <1 empty item> ],
    length: 1,
    red: null
  }
}
truffle(development)> app.candidates(4)
Result {
  '0': BN {
    negative: 0,
    words: [ 4, <1 empty item> ],
    length: 1,
    red: null
  },
  '1': 'Candidate 4',
  '2': BN {
    negative: 0,
    words: [ 0, <1 empty item> ],
    length: 1,
    red: null
  },
  id: BN {
    negative: 0,
    words: [ 4, <1 empty item> ],
    length: 1,
    red: null
  },
  name: 'Candidate 4',
  CountOfVotes: BN {
    negative: 0,
    words: [ 0, <1 empty item> ],
    length: 1,
    red: null
  }
}
truffle(development)>
```

and next via the command: Election.deployed().then(function(i) app=i;) we get a deployed instance of the smart contract in order to assign it to a variable. Then, we just add a reference of each candidate and we use a key(for instance i in order to get this out of the mapping) (here we use as a max number of five candidates) with the commands:

app.candidates(1)

app.candidates(2)

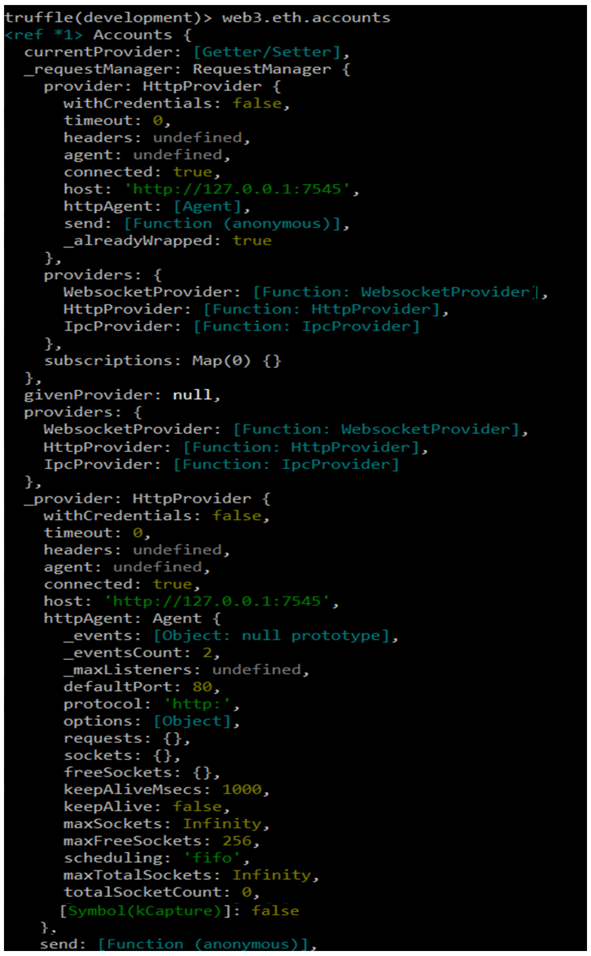
...

app.candidates(5) as seen from above.

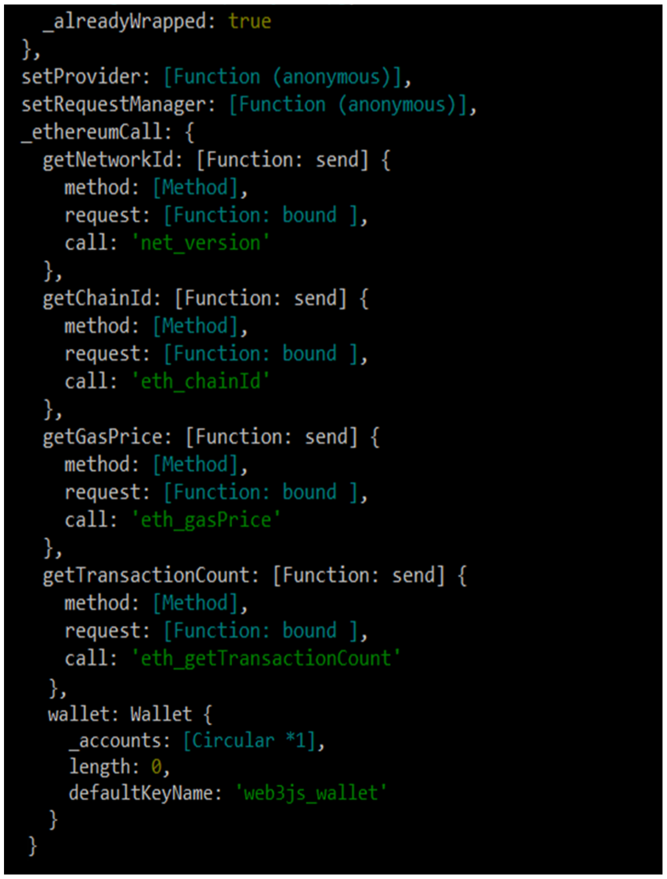
Now, we still need to utilize the added candidates in our smart contract (starting from the first one and then using the first one as the starting point via our code the other four are automatically being utilized through the following command) and in order to do that we use the command:

app.candidates(1).then(function(c) candidates = c;)

then so as to ensure that the candidates have been added successfully to our election system and with the command candidates.id someone is able to see the details of the candidates. Furthermore, with use of the command web3.eth.accounts we can see the available accounts that can be used for voting(for signing transactions and data) in our local Blockchain, in Ganache.

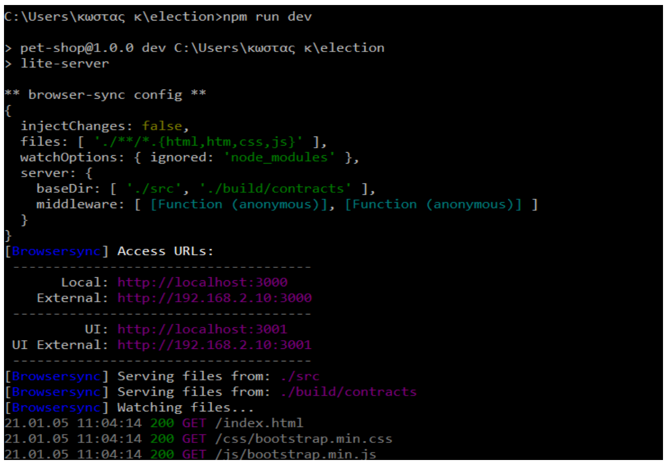


```
truffle(development)> web3.eth.accounts
<ref *1> Accounts {
  currentProvider: [Getter/Setter],
  _requestManager: RequestManager {
    provider: HttpProvider {
      withCredentials: false,
      timeout: 0,
      headers: undefined,
      agent: undefined,
      connected: true,
      host: 'http://127.0.0.1:7545',
      httpAgent: [Agent],
      send: [Function (anonymous)],
      _alreadyWrapped: true
    },
    providers: {
      WebsocketProvider: [Function: WebsocketProvider],
      HttpProvider: [Function: HttpProvider],
      IpcProvider: [Function: IpcProvider]
    },
    subscriptions: Map(0) {}
  },
  givenProvider: null,
  providers: {
    WebsocketProvider: [Function: WebsocketProvider],
    HttpProvider: [Function: HttpProvider],
    IpcProvider: [Function: IpcProvider]
  },
  _provider: HttpProvider {
    withCredentials: false,
    timeout: 0,
    headers: undefined,
    agent: undefined,
    connected: true,
    host: 'http://127.0.0.1:7545',
    httpAgent: Agent {
      _events: [Object: null prototype],
      _eventsCount: 2,
      _maxListeners: undefined,
      defaultPort: 80,
      protocol: 'http:',
      options: [Object],
      requests: {},
      sockets: {},
      freeSockets: {},
      keepAliveMsecs: 1000,
      keepAlive: false,
      maxSockets: Infinity,
      maxFreeSockets: 256,
      scheduling: 'fifo',
      maxTotalSockets: Infinity,
      totalSocketCount: 0,
      [Symbol(kCapture)]: false
    },
    send: [Function (anonymous)],
```

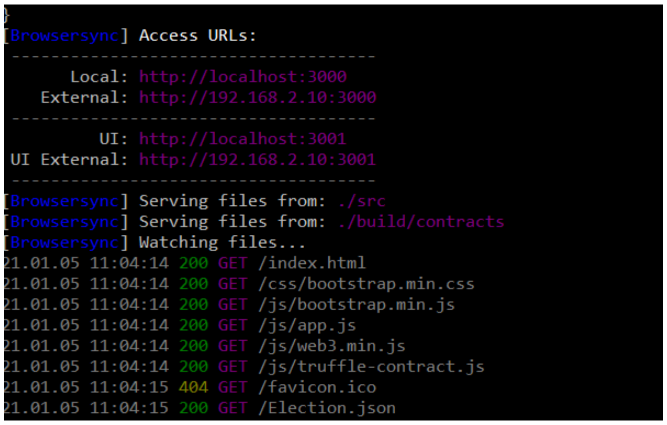


```
    _alreadyWrapped: true
  },
  setProvider: [Function (anonymous)],
  setRequestManager: [Function (anonymous)],
  _ethereumCall: {
    getNetworkId: [Function: send] {
      method: [Method],
      request: [Function: bound ],
      call: 'net_version'
    },
    getChainId: [Function: send] {
      method: [Method],
      request: [Function: bound ],
      call: 'eth_chainId'
    },
    getGasPrice: [Function: send] {
      method: [Method],
      request: [Function: bound ],
      call: 'eth_gasPrice'
    },
    getTransactionCount: [Function: send] {
      method: [Method],
      request: [Function: bound ],
      call: 'eth_getTransactionCount'
    },
    wallet: Wallet {
      _accounts: [Circular *1],
      length: 0,
      defaultKeyName: 'web3js_wallet'
    }
  }
```

Finally, with the use of the command npm run dev the results of our eVoting Decentralized Application are demonstrated in our web browser with our localhost IP: http://127.0.0.1 in port 3000 .

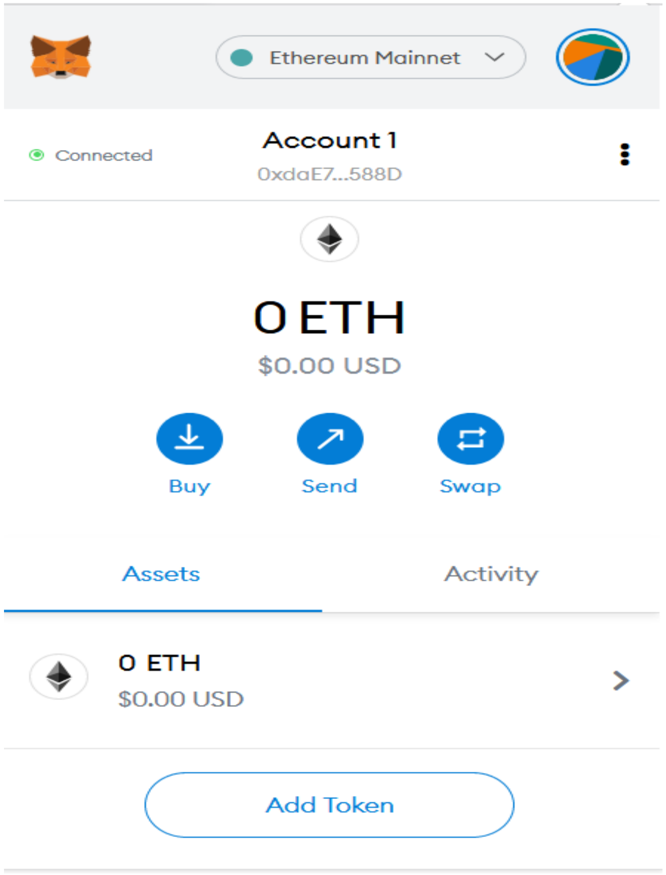


```
C:\Users\κωστας κ\election>npm run dev

> pet-shop@1.0.0 dev C:\Users\κωστας κ\election
> lite-server

** browser-sync config **
{
  injectChanges: false,
  files: [ './**/*.{html,htm,css,js}' ],
  watchOptions: { ignored: 'node_modules' },
  server: {
    baseDir: [ './src', './build/contracts' ],
    middleware: [ [Function (anonymous)], [Function (anonymous)] ]
  }
}
[Browsersync] Access URLs:
 --------------------------------------
       Local: http://localhost:3000
    External: http://192.168.2.10:3000
 --------------------------------------
          UI: http://localhost:3001
 UI External: http://192.168.2.10:3001
 --------------------------------------
[Browsersync] Serving files from: ./src
[Browsersync] Serving files from: ./build/contracts
[Browsersync] Watching files...
21.01.05 11:04:14 200 GET /index.html
21.01.05 11:04:14 200 GET /css/bootstrap.min.css
21.01.05 11:04:14 200 GET /js/bootstrap.min.js
```

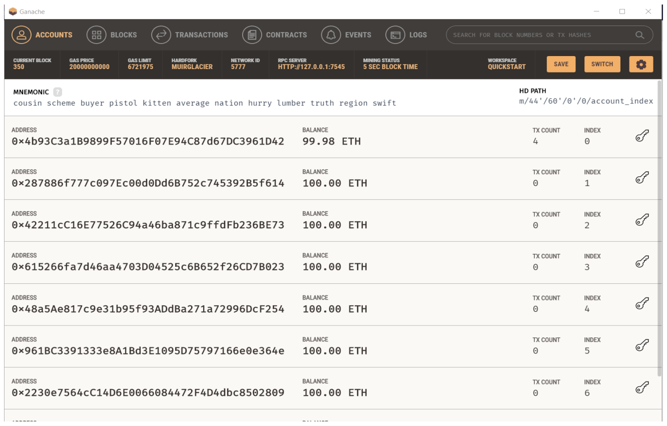

```
}
[Browsersync] Access URLs:
 ------------------------------------
       Local: http://localhost:3000
    External: http://192.168.2.10:3000
 ------------------------------------
          UI: http://localhost:3001
 UI External: http://192.168.2.10:3001
 ------------------------------------
[Browsersync] Serving files from: ./src
[Browsersync] Serving files from: ./build/contracts
[Browsersync] Watching files...
21.01.05 11:04:14 200 GET /index.html
21.01.05 11:04:14 200 GET /css/bootstrap.min.css
21.01.05 11:04:14 200 GET /js/bootstrap.min.js
21.01.05 11:04:14 200 GET /js/app.js
21.01.05 11:04:14 200 GET /js/web3.min.js
21.01.05 11:04:14 200 GET /js/truffle-contract.js
21.01.05 11:04:15 404 GET /favicon.ico
21.01.05 11:04:15 200 GET /Election.json
```

6.4 Setting Up the Connection with Ganache through Metamask

Now that we are in our browser, Metamask is needed to be installed as an extension to the browser that we use. After completing this procedure, we open the Metamask, create an account which is, by default, connected to Ethereum Mainnet.For Developing Testing, we have to change it and connect Metamask with our local Blockchain.



Thereafter, we switch to Ganache application (that we have already pre-launched), we select any address we want and press the key icon of this address.



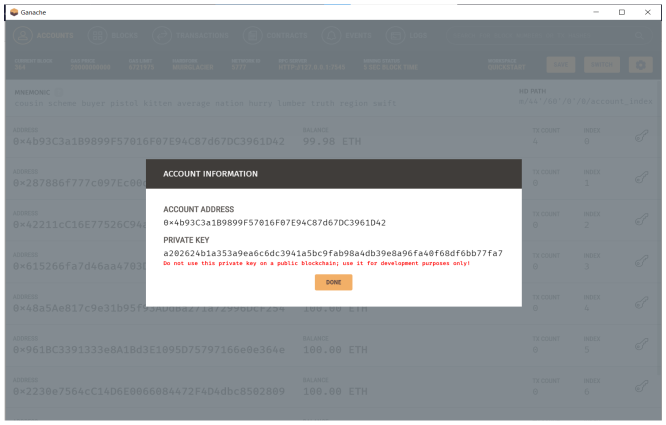
CURRENT BLOCK 350 | GAS PRICE 20000000000 | GAS LIMIT 6721975 | HARDFORK MUIRGLACIER | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:7545 | MINING STATUS 5 SEC BLOCK TIME | WORKSPACE QUICKSTART

MNEMONIC
cousin scheme buyer pistol kitten average nation hurry lumber truth region swift

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×4b93C3a1B9899F57016F07E94C87d67DC3961D42	99.98 ETH	4	0
0×287886f777c097Ec00d0Dd6B752c745392B5f614	100.00 ETH	0	1
0×42211cC16E77526C94a46ba871c9ffdFb236BE73	100.00 ETH	0	2
0×615266fa7d46aa4703D04525c6B652f26CD7B023	100.00 ETH	0	3
0×48a5Ae817c9e31b95f93ADdBa271a72996DcF254	100.00 ETH	0	4
0×961BC3391333e8A1Bd3E1095D75797166e0e364e	100.00 ETH	0	5
0×2230e7564cC14D6E0066084472F4D4dbc8502809	100.00 ETH	0	6

When the key icon is pressed, the address number and the private key of this address are displayed. The private key is needed to be copied to our clipboard in order to be pasted in Metamask.

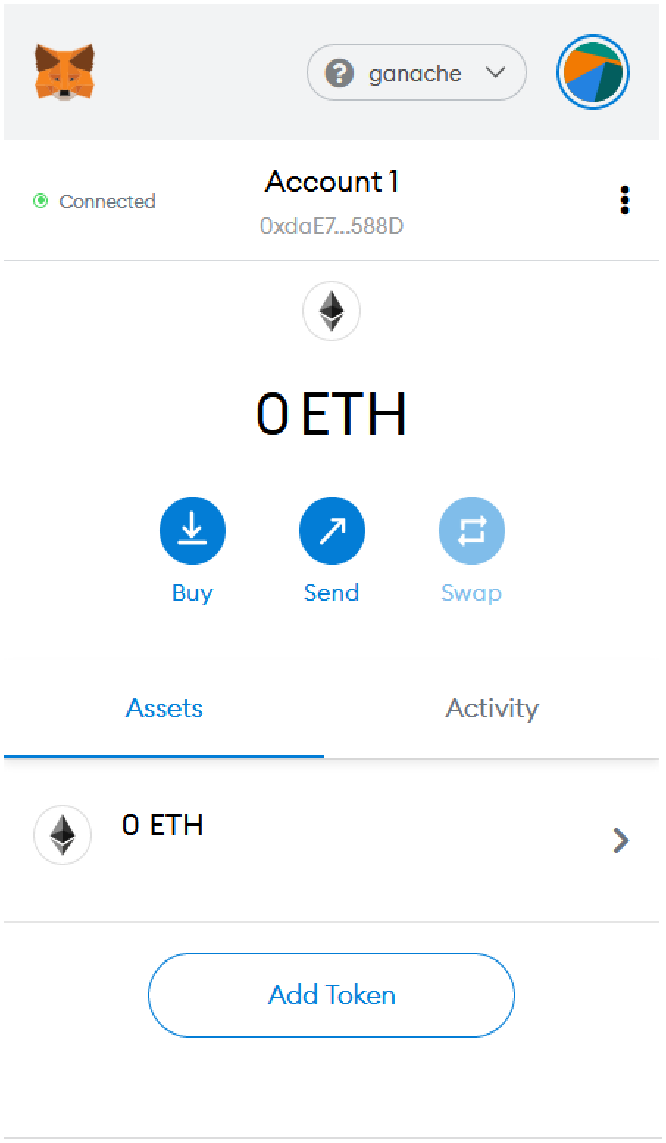


ACCOUNT INFORMATION

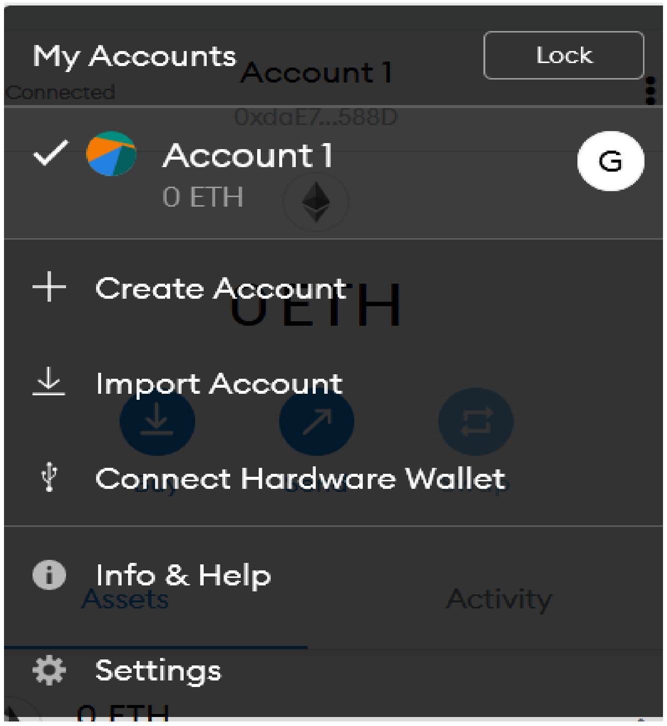
ACCOUNT ADDRESS
0×4b93C3a1B9899F57016F07E94C87d67DC3961D42

PRIVATE KEY
a202624b1a353a9ea6c6dc3941a5bc9fab98a4db39e8a96fa40f68df6bb77fa7
Do not use this private key on a public blockchain; use it for development purposes only!

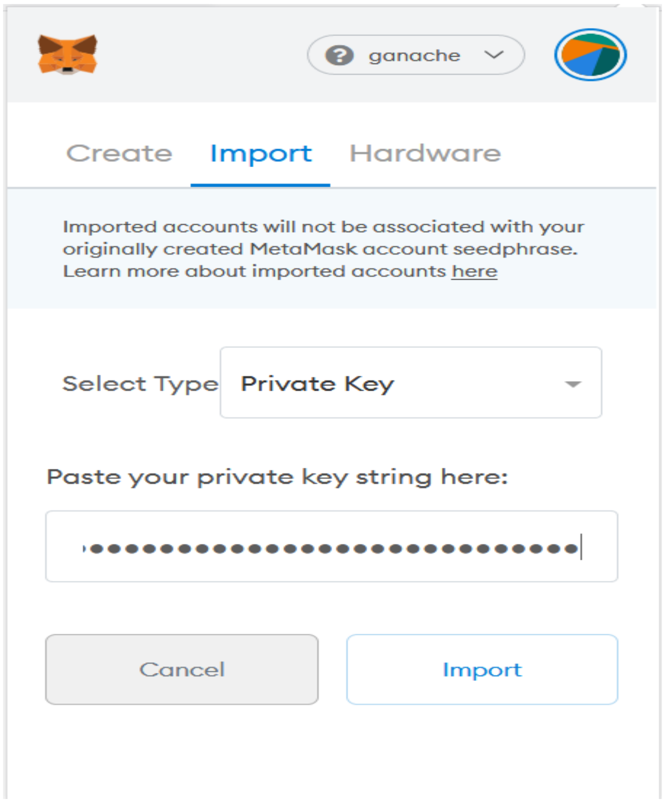
Subsequently, we switch back to Metamask and make sure our current account is connected to our Local Blockchain (Ganache) and not to Ethereum Mainnet anymore.



Now we focus to the row that says Account1 is connected and we press the 3 dots to the right.



This will be shown up and then we select the Import Account selection.

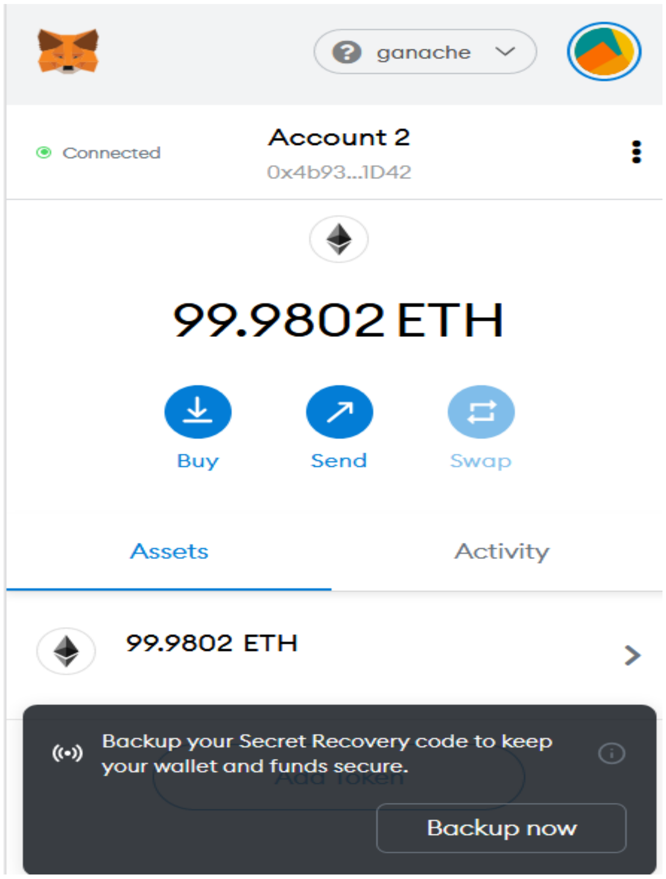


Imported accounts will not be associated with your originally created MetaMask account seedphrase. Learn more about imported accounts here

Select Type: Private Key

Paste your private key string here:

We need to make sure that in Select Type selection we choose Private Key and below we paste the private key that we copied before from Ganache and we press import.

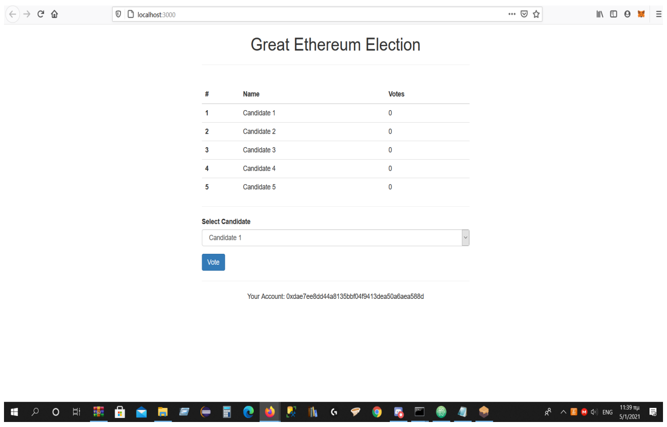


Backup your Secret Recovery code to keep your wallet and funds secure.

6.5 Execution and Display of eVoting Application

As it can been seen Account 2 is connected to Metamask. Now that we imported an account we go back to this address http://localhost:3000 (that is already opened from before when we executed web3) and the election board will be appeared with the candidates' and votes' column and below a small board in order to vote the candidate that you want.

We press confirm and the if we check to our election board the vote is been registered to Candidate 3.

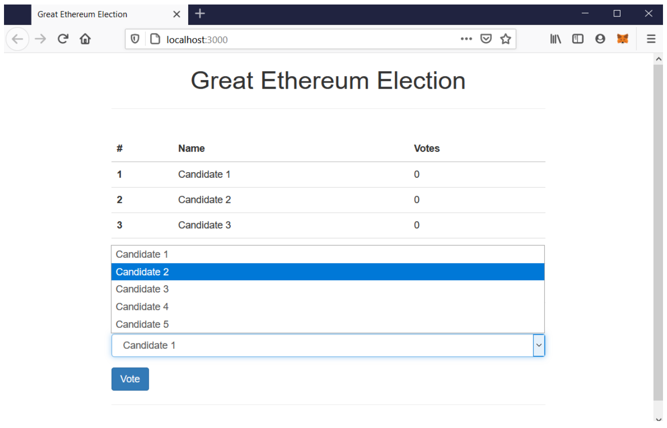


Great Ethereum Election

#	Name	Votes
1	Candidate 1	0
2	Candidate 2	0
3	Candidate 3	0
4	Candidate 4	0
5	Candidate 5	0

Your Account: 0xdae7ee8dd44a8135bbf04f9413dea50a6aea588d

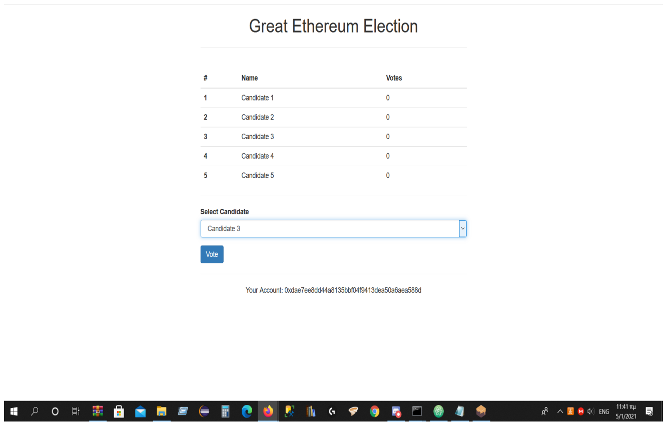
Table 1: Home Screen of eVoting Dapp.



Great Ethereum Election

#	Name	Votes
1	Candidate 1	0
2	Candidate 2	0
3	Candidate 3	0

Table 2: We can choose which Candidate we want to vote.

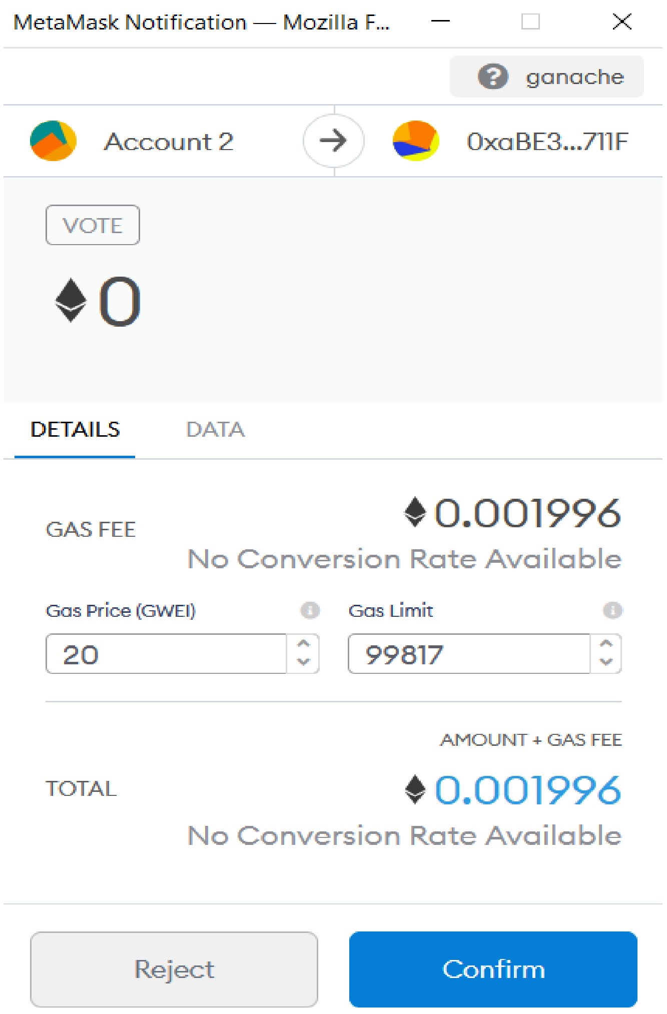


Great Ethereum Election

#	Name	Votes
1	Candidate 1	0
2	Candidate 2	0
3	Candidate 3	0
4	Candidate 4	0
5	Candidate 5	0

Your Account: 0xdae7ee8dd44a8135bbf04f9413dea50a6aea588d

Table 3: For example we chose to vote candidate 3.


MetaMask Notification — Mozilla F...

ganache

Account 2 → 0xaBE3...711F

VOTE

0

DETAILS | DATA

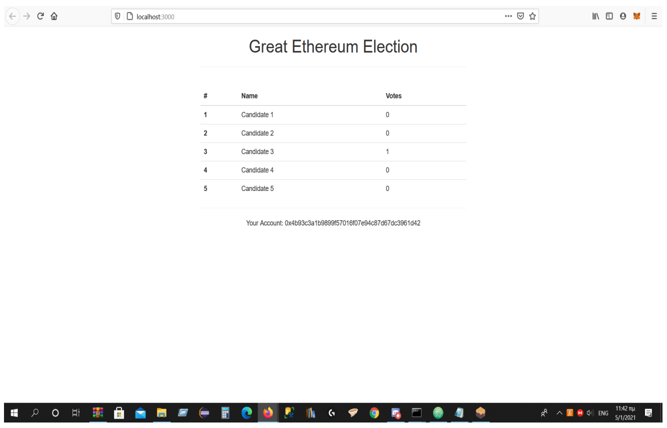
GAS FEE: 0.001996
No Conversion Rate Available

Gas Price (GWEI): 20
Gas Limit: 99817

AMOUNT + GAS FEE
TOTAL: 0.001996
No Conversion Rate Available

Reject | Confirm

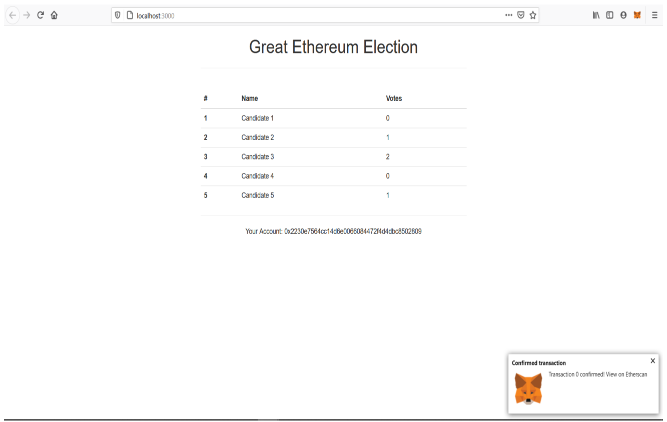
Table 4: Every time we vote, a transaction to the blockchain is actually what is taking place so after we press the vote button, boxes pop up in our screen in order to choose the gas price and the gas limit and also shows up the transaction fee.


localhost:3000

Great Ethereum Election

#	Name	Votes
1	Candidate 1	0
2	Candidate 2	0
3	Candidate 3	1
4	Candidate 4	0
5	Candidate 5	0

Your Account: 0x4b93c3a1b9899f57016f07e94c87d67dc3961d42


localhost:3000

Great Ethereum Election

#	Name	Votes
1	Candidate 1	0
2	Candidate 2	1
3	Candidate 3	2
4	Candidate 4	0
5	Candidate 5	1

Your Account: 0x2230e7564cc14d6e0066084472f4d4dbc8502809

Confirmed transaction
Transaction 0 confirmed! View on Etherscan

Table 5: Results of the Election.

Every time we need to register another vote it is needed to make all the process with Metamask again because we cannot register another vote using the same Metamask account!! For example, repeating the same process we have voted the 2nd, the 5th and once more the 3rd candidate, as it can be seen from the Table 5.

7 CONCLUSIONS

All in all, we can conclude that this innovating technology, Blockchain, has many capabilities and thus it will bring about a great change in many aspects of our everyday life, some of them being more complex and stronger security systems, great medical applications and of course a more trustworthy and fair political system. Namely, as far as the political field is concerned, in the near future there are going to be list insertions with acceptable addresses, that have voting rights and they will be connected with real persons' identification cards. However, a possible security issue might be the existence of fake addresses linked to fake profiles. Nonetheless, this is something that can be changed through the contribution of many experts in security and if measures are taken swiftly then this will be a new beginning for the way of our life works.

7.1 Find our Code

eVoting Decentralized Application source code can be found in Github :
https://github.com/sv1sjp/eVoting_Elections_Decentralized_App/ .
You can try to run it in your computer. Read the instructions above.Moreover, read the README.md for some extra instructions.

REFERENCES

[1] Antonopoulos, Andreas M.; Wood, Gavin (2018). Mastering Ethereum : building smart contracts and DApps (First ed.). Sebastopol, CA: O'Reilly Media, Inc.
[2] "White Paper· ethereum/wiki Wiki · GitHub". Archived from the original on 11 January 2014.
[3] Generalised Transaction Ledger (EIP-150)". yellowpaper.io. Archived from the original on 3 February 2018. Retrieved 3 February 2018
[4] Vitalik Buterin. "Merkling in Ethereum". Ethereum.org.
[5] The Truth Machine: The Blockchain and the Future of Everything - Michael Casey and Paul Vigna
[6] https://en.wikipedia.org/wiki/Ethereum "Ethereum"
[7] https://en.wikipedia.org/wiki/Blockchain "Blockchain"

[8] https://en.wikipedia.org/wiki/Bitcoin"Bitcoin" "Bitcoin"
[9] https://en.wikipedia.org/wiki/Smart_contract "Smart Contract"
[10] Bitcoin: A Peer-to-Peer Electronic Cash System - Satoshi Nakamoto - https://bitcoin.org/bitcoin.pdf
[11] Dapp University - https://www.dappuniversity.com/
[12] https://www.bitdegree.org/courses/solidity-smart-contract/ Basics Of Solidity
[13] Majority is not Enough:Bitcoin Mining is Vulnerable - Ittay Eyal and Emin Gun Sirer : https://arxiv.org/pdf/1311.0243.pdf
[14] ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER - DR. GAVIN WOOD : http://gavwood.com/paper.pdf
[15] Formal Analysis of a Proof-of-Stake Blockchain - Naipeng Dong : https://www.researchgate.net/publication/330030317_Formal_Analysis_of_a_Proof-of-Stake_Blockchain
[16] Proof of Work and Proof of Stakeconsensus protocols: a blockchain applicationfor local complementary currencies - Sothearath SEANG Dominique TORRE : https://gdre-scpo-aix.sciencesconf.org/195470/document
[17] Solidity 0.8.0 Documentation : https://docs.soliditylang.org/en/v0.8.0/
[18] Solidity Github Documentation and examples: https://github.com/ethereum
[19] web3.js Documentation: https://web3js.readthedocs.io/en/v1.3.0/
[20] Truffle Documentaton: https://www.trufflesuite.com/docs/truffle/overview
[21] Ganache Documentation: https://www.trufflesuite.com/docs/ganache/overview
[22] Metamask Guide: https://docs.metamask.io/guide/