Eötvös Loránd University

Faculty of Informatics

Department of Algorithms and their applications

Decentralized voting application based on Ethereum blockchain

Supervisor:
Szabó László
Associate Professor

Author:
Khaligova Jamala
Computer Science BSc

Budapest, 2021

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

Thesis Registration Form

Student's Data:

Student's Name: Khaligova Jamala

Student's Neptun code: A0JCKR

Course Data:

Student's Major: Computer Science BSc

I have an internal supervisor

Internal Supervisor's Name: *Szabó László*

Supervisor's Home Institution: *Eötvös Loránd University, Faculty of Informatics, Department of Algorithms*

Address of Supervisor's Home Institution: *1117 Budapest, Pázmány Péter sétány 1/C*

Supervisor's Position and Degree: *Associate Professor, PhD with Habilitation*

Thesis Title: Secure e-voting

Topic of the Thesis:

(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis.)

One of the main concerns of modern era is transparency of elections. There are many problems related to current election systems: vote rigging, EVM (electronic voting machine) hacks, lacking transparency etc. Blockchain based e-voting system offers secure, trustworthy voting schemes and provides transparency. The data in blockchains are decentralized, meaning that no single server is the center of truth, but the data is synched across hundreds of individual nodes, all of whom are storing the data cryptographically to ensure security and consistency of the data. The data is immutable, and changes are rejected by the majority, something referred to as the "trust protocol". Smart contract is the programming unit of the blockchain. Through smart contracts, data can be written to the blockchain, and the currency of the blockchain can be moved (e.g., $ETH), provided the appropriate conditions are met. Smart contracts are deployed in a singular address in the blockchain, meaning the code that powers applications is decentralized and cannot be altered or changed without re-deploying to a new address. The idea of adapting e-voting systems, to make electoral method cheaper, quicker, and easier, could be compelling in the society. E-voting system consists of pre-election registration process, voting, auditing, counting, and final election results.

I am going to test the efficacy and security of building a proof of concept application comprising of a smart contract to enable voting, and a simple UI to interact with it. Smart contracts will be written in Solidity programming language, collection of code and data that is stored at a specific address on the Ethereum blockchain. Web3.js will be used to interact with the Blockchain, such as making transactions and calls to smart contracts. With the use of Truffle, smart contracts can be compiled, tested, and deployed to Blockchain. Metamask is a browser extension allowing to use decentralized applications.

Budapest, 2021. 09. 07.

Contents

Chapter 1

Introduction

1.1 Motivation

To reflect citizens' opinions on national matters through free and fair elections is necessary in every democratic country. The use of electronic devices in the voting fastened the process, which to some extent prevented the loss of time and human error, but it wasn't completely safe. For example: lack of transparency and vote rigging in Electronic Voting Machines (EVMs) [1], Problems with handling computer cards and vote rigging in Punch cards [2], Hack attacks, bugs, misconfiguration of the software in Ballot-marking devices and systems (BMDs) [3] etc. EVMs and other mentioned centralized online voting devices do not actually solve one of the major issues facing democracies: ensuring trust in the election authority. [4]

Blockchain uses distributed ledger technology (DLT) which provides decentralized system. Entries in DLT are immutable, no one can insert, change, or alter transactions in already validated block. [5] By using distributed database of blockchain, data is shared among all nodes. This eliminates the need of one center that controls everything, therefore provides transparency. Blockchain's identity verification tool can reduce fraud significantly, which makes it more secure. [6] Several types of malicious attacks and many risks can be prevented by using blockchain such as double-spending and record-hacking. On the other hand, the technology is not resistant to some malicious activities such as 51% attack. [7] Currently, there are studies to provide security remediation against such malicious attacks. [8]

In this thesis work, blockchain based electronic voting system called Secure eVote is proposed. How blockchain technology can be used in voting systems, and what

kind of problems it can solve will be investigated.

1.2 Blockchain Technology

Blockchain gained popularity when the paper entitled "Bitcoin: Peer-to-Peer Electronic Cash System" was published by Satoshi Nakamoto in 2008. [9] This paper introduced new way of sending electronic cash from one party to another without the involvement of central authority. The underlying technology that provides this peer-to-peer network is blockchain. This technology is called blockchain, because of the way it stores transaction data — in blocks that are linked together to form a chain.

1.2.1 Structure

The blockchain is the decentralized transparent ledger with the database that is shared by all network nodes, updated by miners, monitored by everyone, and owned and controlled by no one. Its distributed database keeps records of transactions across a P2P network. In client server network, client nodes request for data and server node provides it, while in P2P network each node can be either client or server, meaning that all nodes can request and send data. P2P provides decentralization for blockchain, this allows data to be transferred without the need of central server or intermediaries. Peer-to-peer network in blockchain means that all computers in the network are connected, where each stores complete copy of the ledger and compares it to other devices to ensure the data is accurate.

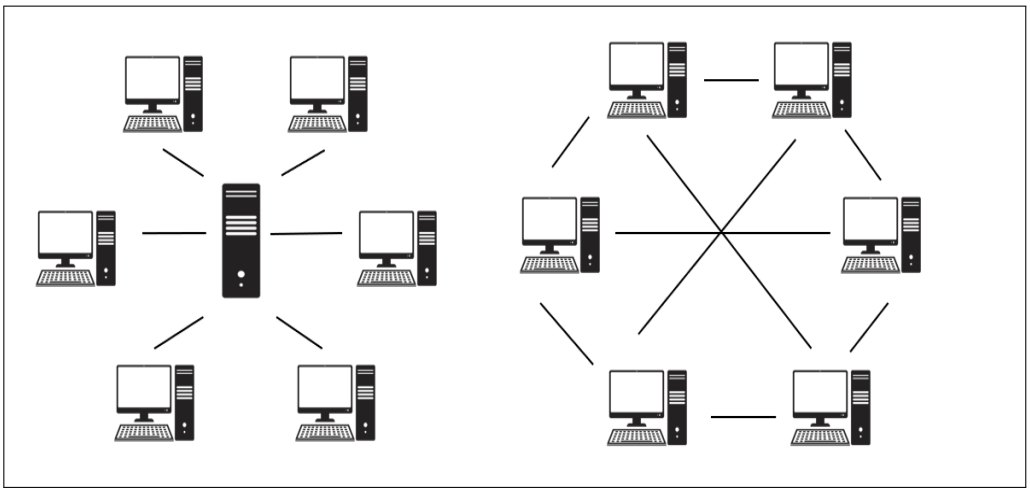

Figure 1.1: Client Server vs P2P network

Each block in blockchain contains block header and transaction data. Block header contains metadata of the block, six components which are the version, hash of the previous last block, root hash of the Merkle tree, time stamp, the goal of the current difficulty, nonce. [10] With version number, miners can track any changes or upgrades made in the protocol. The primary identifier of each individual block is the cryptographic hash it contains. Each block carries the hash of the block that was mined before it. This renders immutability to the blocks. Each block contains a hash reference to previous block, known as parent block. The sequence of hashes linking each block to its parent creates a chain going back all the way to the first block ever created, known as the genesis block.

The "previous block hash" field is inside the block header and thereby affects the current block's hash. If parent's identity changes, then child's identity also changes. Any modification in parent causes its hash to change. The parent's changed hash necessitates a change in the "previous block hash" pointer of the child. This in turn causes the child's hash to change, which requires a change in the pointer of the grandchild, which in turn changes the grandchild, and so on. This cascade effect ensures that once a block has many generations following it, it cannot be changed without forcing a recalculation of all subsequent blocks. Because such a recalculation would require enormous computation, the existence of a long chain of blocks makes the blockchain's deep history immutable, which is a key feature of bitcoin's security. [11]

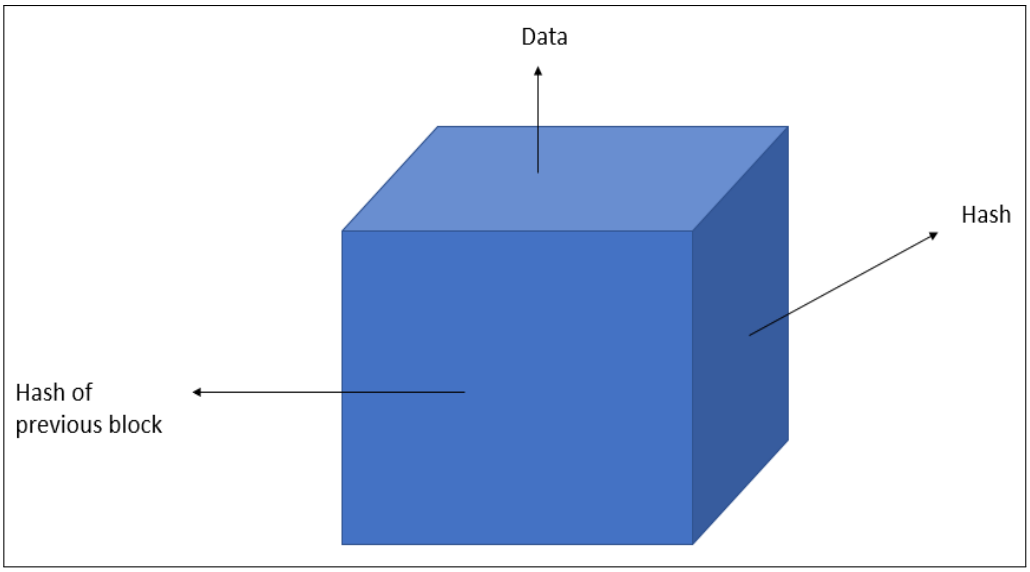

Figure 1.2: Block's basic visualization

A Merkle root is the hash of all the hashes of all the transactions that are part of a block in the blockchain network. It is described in more detail in the next subsection. Difficulty target is a measure of how hard is it for miners working to solve the block. A high cryptocurrency difficulty means it takes additional computing power to verify transactions entered on a blockchain. The nonce is the number that blockchain miners are solving for, in order to receive cryptocurrency, this number is used only once. [12]

Block Content	
Magic Number	4 bytes
Block size	4 bytes
(Header) Version	4 bytes
(Header) Previous Block Hash	32 bytes
(Header) Merkle Root	32 bytes
(Header) Time Stamp	4 bytes
(Header) Difficulty Target	4 bytes
(Header) Nonce	4 bytes
Transaction Counter	1 to 9 bytes
Transaction	Depends on the transaction size

Table 1.1: Block Structure

1.2.2 Merkle Tree

Merkle Tree is a binary tree data structure used in computer science applications. In simple terms, Merkle tree takes a lot of data, compresses it into a simple string of characters that can prove the accuracy of the data held in it without revealing what it is. In this way, it is guaranteed that the data will not be changed. Merkle trees are formed by repeatedly joining pairs of nodes until only one hash remains. The top hash is more commonly called the Merkle Root or Root Hash. It is created from the hash of each of the transactions, from bottom to the top. Generally, a cryptographic hash function such as SHA-256 is used for hashing. If Merkle trees only need to be protected from unintentional damage, a more standard practice such as CRCs is used.

The Merkle tree provides efficient and secure validation of big data structures. The main use of Merkle trees today is to check that data blocks from various nodes in a peer-to-peer network are correctly transferred and even check that other peers in the network do not send blocks with fake content. [13]

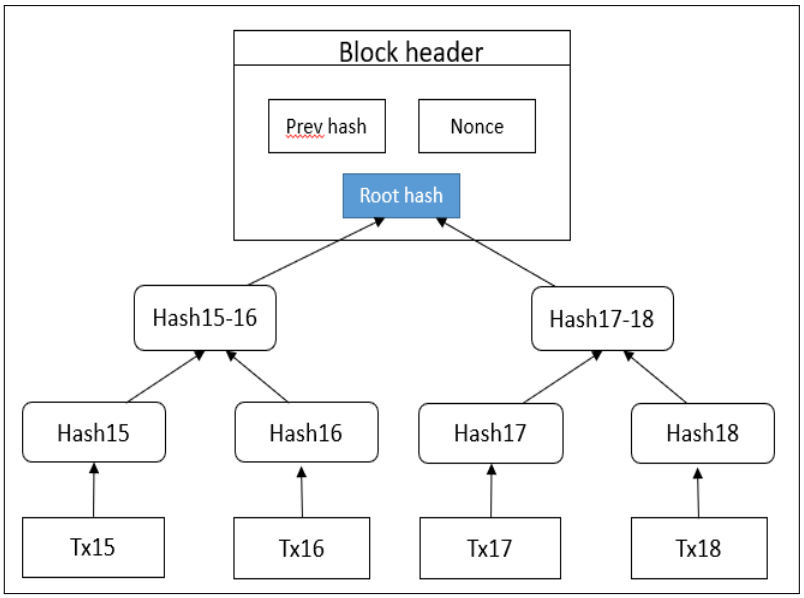

Figure 1.3: Merkle Tree

1.2.3 Proof-of-Work (PoW)

Proof of Work is mainly used in cryptocurrency mining, it rewards miner for solving complex mathematical equations. To solve these mathematical equations requires a lot of computational power, for example, hash functions and integer factorization. The most popular application of PoW is Bitcoin. Cryptocurrencies requires PoW, because it provides security and consensus and eliminates the need of third party.

PoW is an algorithm which confirms the transaction and creates a new block to the chain. In this algorithm, multiple miners are competing against to solve proof of work. This process is called mining. As soon as miners solve the puzzle, new block is formed. Then winner miner is rewarded with cryptocurrency. In general, the miner with more computing power may be an early winner. Determining which string to use as the nonce, meaning that solving proof of work, requires a significant amount of trial-and-error, as it is a random string. A miner must guess a nonce, append it to the hash of the current header, rehash the value, and compare this to the target hash. If the resulting hash value meets the requirements, the miner has created a solution and becomes the winner miner. Finding the nonce on the first try is almost impossible, usually the miner starts with Nonce value of 0 and keeps incrementing it until it gets it right. It may take large amount of iterations to generate the desired hash. Usually, it takes 10 minutes for miners to generate a block. The nodes that receive the new block will accept it only after verifying that all transactions in the

block are valid and not already spent. [14]

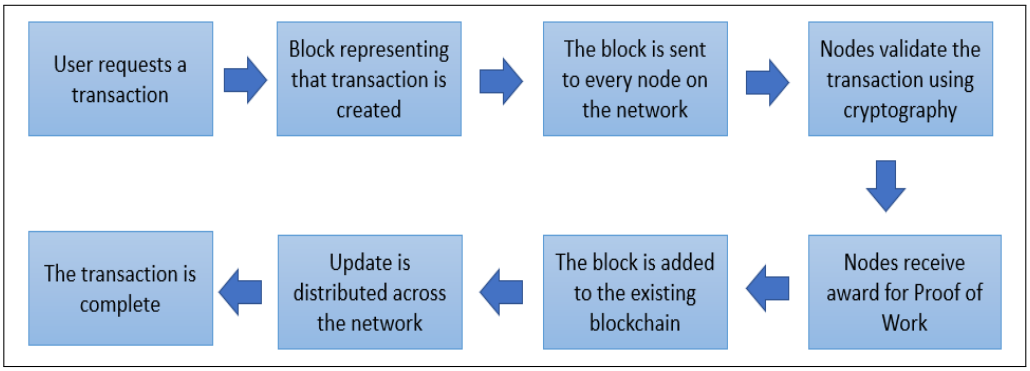

Figure 1.4: How Blockchain Works - PoW

1.2.4 Proof-of-Stake (PoS)

PoS relies on the validators to maintain the cryptocurrency. PoS algorithm chooses the validator of the next block randomly, factors affect selection such as, staking age, randomization, and node's wealth. Each proof-of-stake protocol works differently in how it chooses validators.

Users are required to stake their cryptocurrencies to become a participant for the mining (forging) process. Chances for being a validator are high, if user is staking higher amount. To ensure fairness, other methods are added to selection process, which are "Randomized Block Selection" and "Coin Age Selection". To summarize these methods, the validator is determined in the Randomized Block Selection by the lowest hash value and the highest stake, and in the Coin Age Selection by the duration of days their tokens are staked for. Unlike proof-of-work, validators don't need to use significant amounts of computational power because they're selected at random and aren't competing. When a node gets chosen to mine the next block, it will check if the transactions in the block are valid, signs the block and adds it to the blockchain. As a reward, the node receives the transaction fees that are associated with the transactions in the block. [15]

The main advantages of the Proof of Stake algorithm are energy efficiency and security. Also, randomization makes the network more decentralized. Proof of Stake (PoS) is seen as less risky in terms of the potential for miners to attack the network, as it structures compensation in a way that makes an attack less advantageous for the miner. [16]

1.2.5 SHA-256

SHA-256 code stands for Secure Hash Algorithm. In this algorithm, hash of the data is irreversible and unique, which means that if we produce hash of a certain data multiple times with the same hashing algorithm, the resulting hash would be the same, and given the hash you cannot obtain the original data.This algorithm was developed by the US National Security Agency (NSA), which specializes in cryptology. SHA-256 is used in proof-of-work calculations in bitcoin mining and bitcoin address generation. SHA-256 is a part of the SHA-2 family of algorithms, which is famous for high security and speed. [17]

SHA-256 has been adopted by a number of different blockchain projects, including several coins created from forks of the original Bitcoin source code. Among the top three SHA-256 blockchain projects by market capitalization— Bitcoin (BTC), Bitcoin Cash (BCH), and Bitcoin Satoshi's Vision (BSV). SHA-256 is deterministic algorithm, meaning that it always produces the same output given the same input. It is not possible to reveal input from generated 256 bits hash output. SHA-256 is computationally efficient and an ordinary computer can perform the operation dozens or even hundreds of times per second. The SHA-256 algorithm is important because it's an integral part of mining on the Bitcoin network, as well as many smaller Proof of Work blockchain networks. In basic terms, the SHA 256 hash supports a Proof of Work network in which computers race to solve a complicated math problem. After one computer finds a solution, it broadcasts that solution to the rest of the computers on the peer to peer network. This proves their work to the other machines that were trying to solve the same problem, as each computer on the network verifies the solution independently.[18]

1.2.6 Types of blockchain

Public blockchain is the first type of blockchain technology. Public blockchains are open to the public and any individual can involve in the decision-making process by becoming a node, but users may or may not be benefited for their involvement in the decision-making process. No one in the network has ownership of the ledgers and are publicly open to anyone participated in the network. Each user in the network stores copy of ledger on their local nodes. All transactions that take place on public blockchains are fully transparent, meaning that anyone can examine the transaction

details. Two most popular examples for public blockchain are Bitcoin and Ethereum. [19]

Private blockchain works in a restrictive environment, it is also known as permissioned blockchain. Permissioned blockchains tend to be more efficient. Because access to the network is restricted, there are fewer nodes on the blockchain, resulting in less processing time per transaction. This type of blockchain is controlled by central authority, so which nodes will have access to the blockchain can be decided by authorized nodes. This makes private blockchain more centralized and ideal for auditing, trade secret management, and asset ownership. Examples of private blockchains are Multichain, Hyperledger projects, etc.

Hybrid blockchain combines the privacy benefits of a permissioned and private blockchain with the security and transparency benefits of a public blockchain. That gives businesses significant flexibility to choose what data they want to make public and transparent and what data they want to keep private. Examples of hybrid blockchains are Xinfin, Ripple, etc.

Consortium blockchain is also permissioned platform, like private blockchain, it is controlled by group of people, but works across different organizations. This eliminates the risks that come with one entity controlling the network. These blockchains are more scalable and safer than public blockchains. Consortium blockchains are mainly used by banks, government organizations, etc. [20]

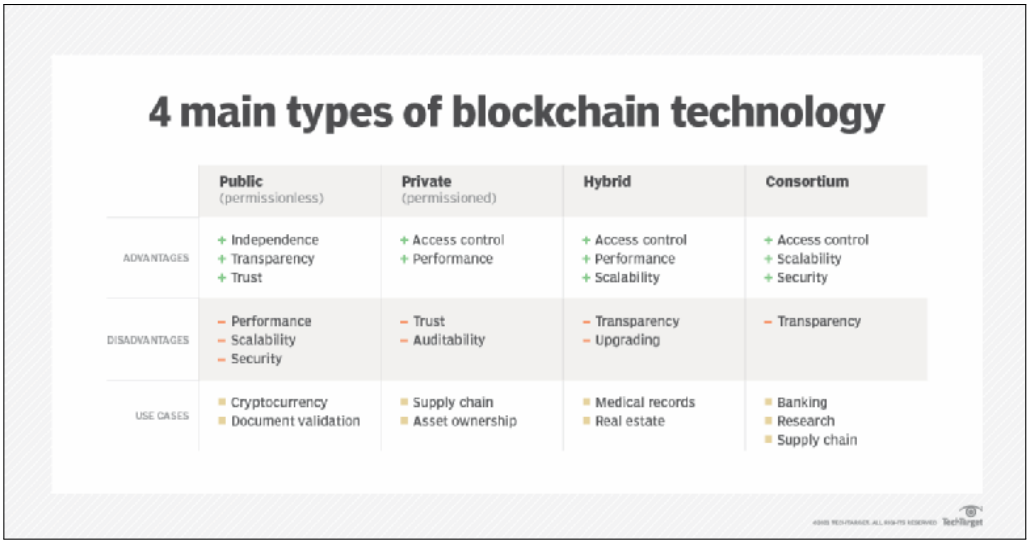

Figure 1.5: Blockchain Types
[19]

1.2.7 Applications of Blockchain

Although blockchain initially developed as underlying technology for Bitcoin, it has the potential to be used in various fields, such as finance, healthcare, real estate,supply chain, IoT, and politics. Blockchain approach becomes advantageous in application areas where it is difficult to provide control and security from a single source, and the cost of establishing a distributed and reliable center is high.

Blockchain has been adopted by many international institutions in a short time, as it supports transparent, secure, fast and efficient operation of business in financial institutions. The advantages that arise as a result of the relationship between blockchain and finance can be listed as follows: increased transparency, improved capital optimization due to elimination of intermediaries, better performance, etc.

In the field of healthcare, blockchain can be used to keep medical records of patients, to transfer them safely between parties, to detect and prevent drug fraud. Blockchain technology is thought to be the cure for these problems, and there are large investments for Blockchain applications in the field of health. The Estonian government started a project called Guardtime in 2011 and made the healthcare platform work on blockchain technology. [21]

Traditional real estate technology deals with connecting buyers and sellers. With Real Estate technology, which will be transformed by blockchain technology, it has been understood that buyers and sellers can perform their transactions without intermediaries such as lawyers, brokers, and banks. The ATLANTA project is an example of blockchain based real estate application. [22] The use of blockchain technology in the real estate field offers advantages such as more secure way to transfer ownership, reduce paperwork and cost, speed up transactions.

In supply chain management, transactions can be carried out on a common blockchain and payments after delivery can be automated without the need for the approval of a reliable center. The follow-up of these transactions can be monitored transparently by the parties at every stage with blockchain technology. It is not possible for one of the parties in the system to delete the records or change them retrospectively. [23]

The blockchain structure ensures the immutability of the records keeping the history of transactions made by IoT devices. In this way, devices will have the ability to interact with each other by using these records as a verification tool without the

need for a single central authority. This technology can be solution for security issues of IoT. This system can provide significant savings in the industry by using it for tracking billions of connected devices, activating transactions and providing coordination between devices.

Blockchain technology can even help optimize the democratic process. Counting and tracking votes can be difficult with traditional methods, which creates an obstacle to democracy. Voting with the help of blockchain can make elections more stable and measurable. Transparent, immutable and decentralized features of blockchain could solve many problems that are happening in governments.

1.2.8 Bitcoin

The cryptocurrency known as Bitcoin was created as a solution for "Double Spending". Double-spending is a problem that arises when transacting digital currency that involves the same tender being spent multiple times. Multiple transactions sharing the same input broadcasted on the network can be problematic and is a flaw unique to digital currencies. The primary reason for double-spending is that digital currency can be very easily reproduced. [24] Bitcoin's network prevents double-spending by combining security features of the blockchain network and its decentralized network of miners to verify transactions before they are added to the blockchain.

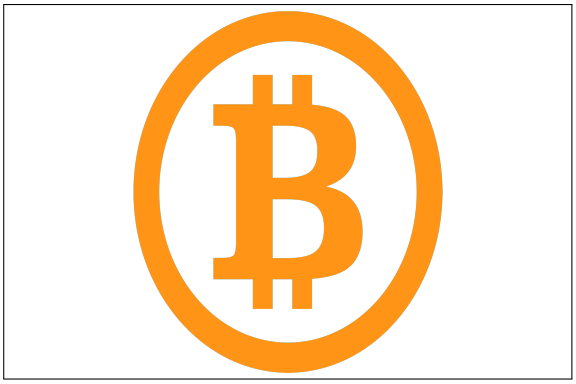

Figure 1.6: Bitcoin logo
[25]

Having blockchain as a base technology, Bitcoin provides its users with secure, transparent, fast and low-cost transfers and transactions. It is not possible to close or control this decentralized system by any person or authority. Users can easily make unlimited amount of transfers and all of them can be clearly observed by other users.

However, the identity of the owner of the wallet to which BTC is transferred is not known unless it is disclosed by him. This ensures the confidentiality of personal data and assets.

Bitcoin is not unlimited, only 21 million units will be produced, and this process is expected to be completed in 2140. However, considering its current position, there is a possibility that this period may be pulled earlier. The limited supply of Bitcoin has led to its comparison with gold and has become a demanded investment tool. This factor plays a big role in the formation of the crypto money market. [26]

1.3 Ethereum

Ethereum is a system that was first introduced at the North American Bitcoin Conference by Ethereum founder Vitalik Buterin. Although it is generally seen as an altcoin, Ethereum is an innovative system that aims to develop blockchain technology and use it in more areas. [27]

Ethereum provides a platform that allows building decentralized applications on its blockchain by executing programs called "smart contracts". Like other blockchains, Ethereum uses a system that is distributed among hundreds of servers around the world. Each node participating in the Ethereum protocol runs the software on its own computer. This vast, decentralized network of computers is what is often referred to as the Ethereum Virtual Machine (EVM). Ethereum Virtual Machine guarantees security for its users by preventing denial-of-service attacks (DDoS), which are malicious actions that aim to shut down a machine or even the entire network and make them completely inaccessible. Another function of the EVM is that it decrypts the passwords created by the Ethereum programming language and ensures that communication can be provided without any interference. Through EVM, this platform allows you to run applications in a distributed way using the language called Solidity. In other words, when you want to perform a transaction mutually, in addition to being able to perform simple transactions as in Bitcoin, it also allows you to program these transactions with certain rules. Every node participating in the network runs the EVM as part of the block verification protocol. They go through the transactions listed in the block they are verifying and run the code as triggered by the transaction within the EVM. [28]

In the Ethereum network, the state is stored by Externally Owned Accounts or EOAs. Each account has a 20-byte address, and transactions take place between these 20-byte addresses. Public-private key pair is generated for each account. While private key is kept secure, public key acts as the identity of the EOA. Accounts are not identified by the person's name or any personal information. An EOA object has several properties:

- A counter value that is kept to ensure that transactions occur only once; nonce.
- The amount of Ether in the account (Ether is Ethereum cryptocurrency).
- The executable contract code, if any.
- Data stored for the account.
- Controlled by private keys.

Contract Accounts are managed by the smart contracts. It holds the amount of ether that it owns. In this account type, the execution of the program is triggered by a transaction or message from other accounts. These accounts are generated when smart contracts deployed to the blockchain.

Operations such as sending money to an address or calling a function of the contract are called transactions. They are signed packets of data, that are triggered by an address on the Ethereum network. These messages include:

- The recipient of the message.
- A digital signature that identifies the sender.
- The amount of Ether to be transmitted to the recipient.
- An optional data storage area.
- STARTGAS value calculated based on the maximum possible processing step calculated by the EVM. This value corresponds to the maximum cost that may arise as a result of the operation of the relevant message.
- GASPRICE, which determines the amount of Ether payable by the sender.

The unit called "gas" is calculated according to each process step. In other words, if too much data is generated in the next contract, then the amount of gas to be paid increases accordingly. This gas value is also sent to the miner running the transaction as a reward for using the processing power. For example, if a smart contract makes an account and transfers Ether or a token, it will create a new state, so a certain amount of gas must be paid for this, while reading the current status of an existing contract will not create a new state, so can happen for free.

1.3.1 Smart Contracts

Smart contracts were first proposed in 1994 by American computer scientist Nick Szabo, who invented a virtual currency called "Bit Gold" in 1998, exactly 10 years before the invention of Bitcoin.

Smart contracts are computer programs that can carry out transactions and agreements between anonymous parties in a reliable and consistent manner and cannot be changed retrospectively.As smart contracts are typically deployed on and secured by blockchain, they have some unique characteristics. First, the program code of a smart contract will be recorded and verified on blockchain, thus making the contract tamper-resistant. Second, the execution of a smart contract is enforced among anonymous, trustless individual nodes without centralized control, and coordination of third-party authorities. Third, a smart contract, like an intelligent agent, might have its own cryptocurrencies or other digital assets, and transfer them when predefined conditions are triggered. [29]

A real world example of the use of smart contracts is in Supply Chain and Tracking, as a distributor you can use smart contracts to sell and distribute your products all over the world. Many aspects of the supply chain can be replaced by a smart contract, making the entire system more efficient and more fraud resistant. Of course, usage of smart contracts is not limited to supply chain only, it can also be used in insurance policies and payments, stock trading, e-voting, auction, and other decentralized applications.

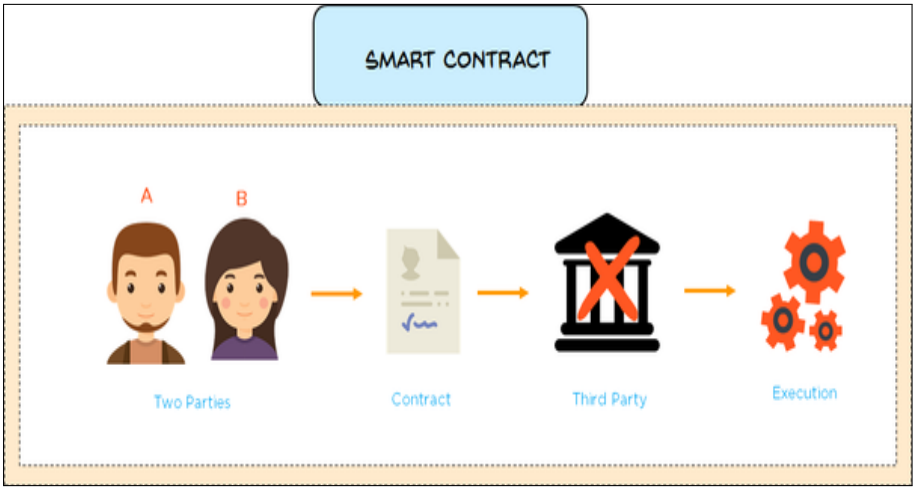

Figure 1.7: Smart Contract
[30]

1.3.2 Solidity

Solidity is a brand-new programming language created by the Ethereum, released in the year 2015 led by Christian Reitwiessner. It is a high-level programming language designed for implementing smart contracts. Solidity is highly influenced by Python, C++, and JavaScript which runs on the Ethereum Virtual Machine(EVM). It is statically-typed object-oriented(contract-oriented) language. Solidity is primary language for blockchains running platforms. Solidity can be used to creating contracts like voting, blind auctions, crowdfunding, multi-signature wallets, etc.[31]

```
1  // Solidity program to demonstrate how to write a smart contract
2  pragma solidity >= 0.4.16 < 0.7.0;
3
4  // Defining a contract
5  contract Test
6  {
7
8      // Declaring state variables
9      uint public var1;
10     uint public var2;
11     uint public sum;
12
13     // Defining public function that sets the value of the state
           variable
14     function set(uint x, uint y) public
15     {
16         var1 = x;
17         var2=y;
18         sum=var1+var2;
19     }
20
21     // Defining function to print the sum of state variables
22     function get(
23     ) public view returns (uint) {
24         return sum;
25     }
26 }
```

Code 1.1: Simple smart contract written in Solidity. [31]

Chapter 2

User documentation

This chapter will guide users how to install and run the application. The technologies used to achieve blockchain based voting application will be summarized briefly. Decentralized voting application provides transparency, reduces fraud, meaning that it solves some of the most important problems currently happening in online voting. Also, test outputs will be included in this chapter to demonstrate what the user should see after running the program.

2.1 Software Installation Guide

To run the application on your device, you need to install listed software programs. Since UI of this application is web based, widely known programming languages such as HTML, CSS, Jquery (JS) were used for the front-end. I will shortly explain the technologies which may be new for the majority and blockchain-specific.

1. NodeJS (npm) installation
2. Truffle installation
3. Ganache installation
4. Web3 introduction
5. Setting up MetaMask

Npm is a package manager for NodeJS modules. It is installed on your computer when you install NodeJS. For this application, you need to download NodeJS v14.17.5, and npm version v7.23.0 will be downloaded automatically. [32] If npm version is different, "npm install npm@7.23.0" is used to install required version.

Code below demonstrates checking and installing correct npm version in Windows Command Prompt:

```
C:\Users\cemal>node -v
v14.17.5

C:\Users\cemal>npm -v
7.23.0

C:\Users\cemal>  npm install npm@7.23.0

added 1 package, and audited 258 packages in 46s
```

Code 2.1: NodeJS and npm version control

Truffle was created in 2015 by Tim Coulter after spending time with Ethereum and ConsenSys trying to build blockchain applications. He started to develop a few scenarios to make his own life easier and help the development process. The result was what is known as the Truffle Suite, which consists of three components that are Truffle, Ganache, and Drizzle. With Truffle, you can compile, deploy and test smart contracts, inject them into web apps, and also develop front-end for DApps. There are some key features makes Truffle powerful tool to build DApps: [33]

- Built-in support to Compile, Deploy and Link smart contracts
- Automated Contract testing
- Supports Console apps as well as Web apps
- Network Management and Package Management
- Truffle console to directly communicate with smart contracts
- Supports tight integration

The configuration file, also known as, truffle.js or truffle-config.js, defines how Truffle can connect to Ethereum networks. [34] Truffle v5.4.8 must be installed to compile, deploy and test smart contracts. If the this version is deprecated at the time you try to install, then please install updated version. The code below shows how to install truffle in Windows Command Prompt:

```
C:\Users\cemal>npm install -g truffle@5.4.8

C:\Users\cemal>truffle -v
Truffle v5.4.8 - a development framework for Ethereum

```

```
Usage: truffle <command> [options]

Commands:
  build     Execute build pipeline (if configuration present)
  compile   Compile contract source files
  config    Set user-level configuration options
  console   Run a console with contract abstractions and commands
      available
  create    Helper to create new contracts, migrations and tests
  db        Database interface commands
  debug     Interactively debug any transaction on the blockchain
  deploy    (alias for migrate)
  develop   Open a console with a local development blockchain
  exec      Execute a JS module within this Truffle environment
  help      List all commands or provide information about a
      specific command
  init      Initialize new and empty Ethereum project
  install   Install a package from the Ethereum Package Registry
  migrate   Run migrations to deploy contracts
  networks  Show addresses for deployed contracts on each network
  obtain    Fetch and cache a specified compiler
  opcode    Print the compiled opcodes for a given contract
  preserve  Save data to decentralized storage platforms like IPFS
      and Filecoin
  publish   Publish a package to the Ethereum Package Registry
  run       Run a third-party command
  test      Run JavaScript and Solidity tests
  unbox     Download a Truffle Box, a pre-built Truffle project
  version   Show version number and exit
  watch     Watch filesystem for changes and rebuild the project
      automatically

See more at http://trufflesuite.com/docs
```

Code 2.2: Truffle installation and commands

Ganache is a personalized blockchain for Ethereum development. It can be used to run tests, execute commands, and inspect states while controlling how the chain operates. Ganache can be installed from this website. [35]

After installation, you need to upload truffle-config.js from the project's files to set up your workspace. The name for the workspace is automatically generated, or

you can change it as you like.

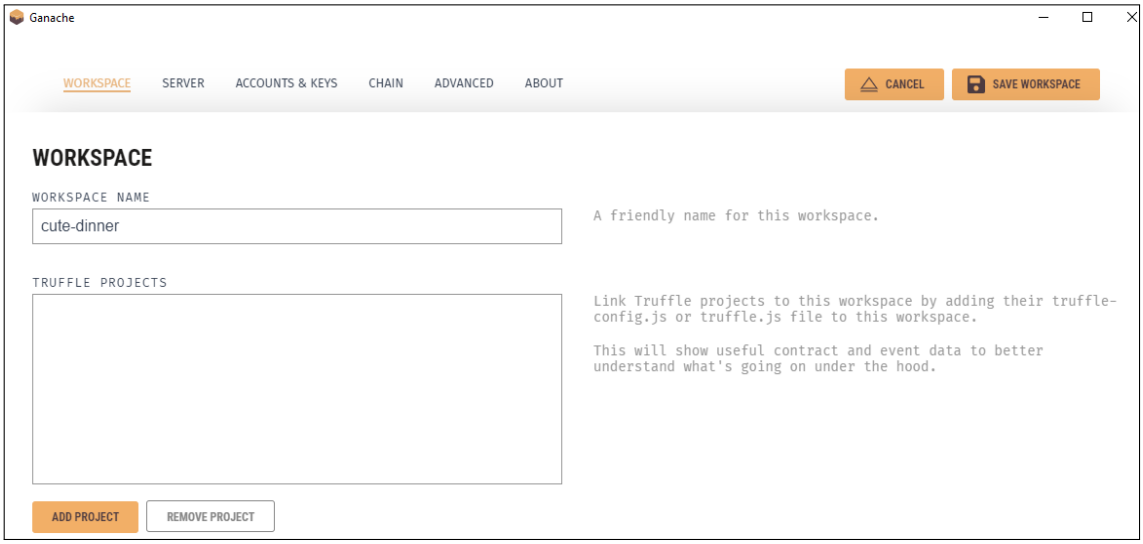

Figure 2.1: Ganache set up workspace

The server port number should be configured as "7545". Because in truffle-config.js, the port is specified the same way.

Web2 refers to the version of the internet most of us know today. An internet dominated by companies that provide services in exchange for your personal data. Web3, in the context of Ethereum, refers to decentralized apps that run on the blockchain. These are apps that allow anyone to participate without monetising their personal data. Many Web3 developers have chosen to build dapps because of Ethereum's inherent decentralization: [36]

- Anyone who is on the network has permission to use the service – or in other words, permission isn't required.
- No one can block you or deny you access to the service.
- Payments are built in via the native token, ether (ETH).
- Ethereum is turing-complete, meaning you can pretty much program anything.

In this documentation, I won't include installation for web3 module, since I already included minified version "web3.min.js" in the project.

MetaMask is an easy-to-use browser plugin (for Google-Chrome, Firefox and Brave browser), that provides a graphical user interface to make Ethereum transactions. It allows you to run Ethereum DApps on your browser without running a full Ethereum node on your system. Basically, MetaMask acts as a bridge between Ethereum Blockchain and the browser. MetaMask is open-source and provides the following exciting features: [33]

- You can change the code of MetaMask to make it what you want it to be.
- Provides built-in coin purchasing.
- Local-key Storage.

To use MetaMask, you need to "add to Chrome" in Chrome web store. [37] Then you need to set up a password for your account. After completing registration steps, your MetaMask wallet will be ready for usage. Then you can navigate to networks, and add new network to use the application in the port 7545.

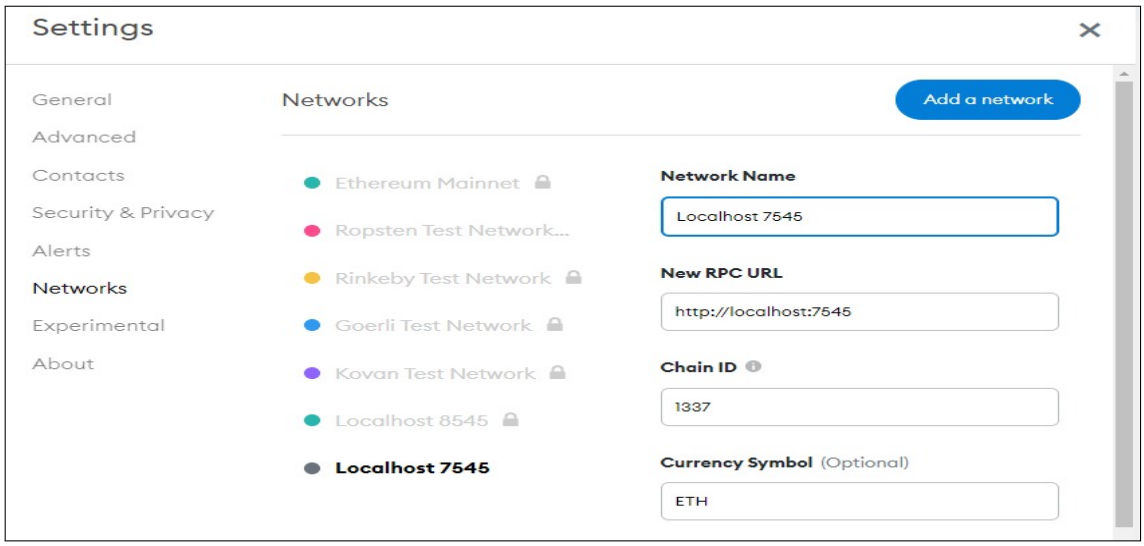

Figure 2.2: MetaMask Networks

Lastly, accounts must be imported from Ganache. To do this, you need to click on the import account in MetaMask, and select type "private key". In Ganache, click on the key picture, displayed on right side for each account address. You need to copy private key from there, and paste in MetaMask. You can see this steps in the following screenshots.

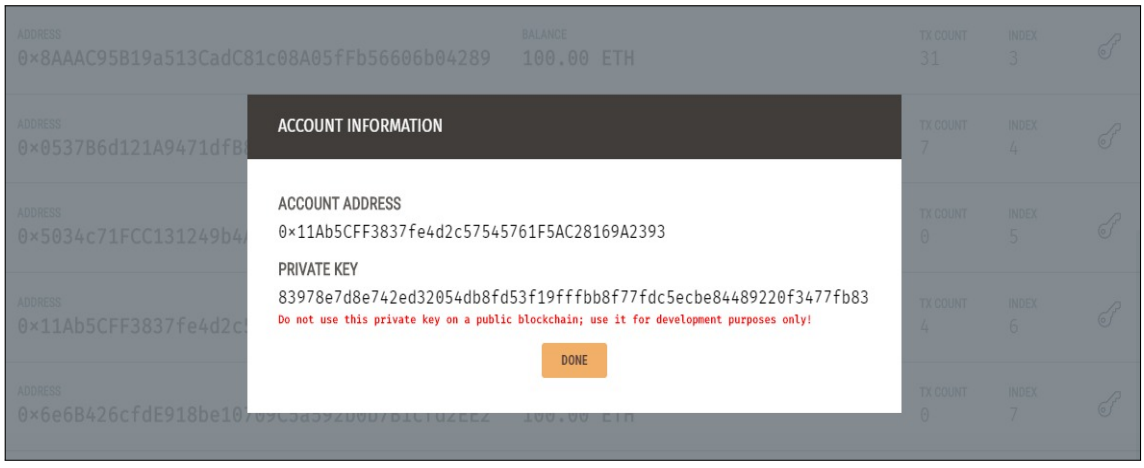

Figure 2.3: Private key in Ganache

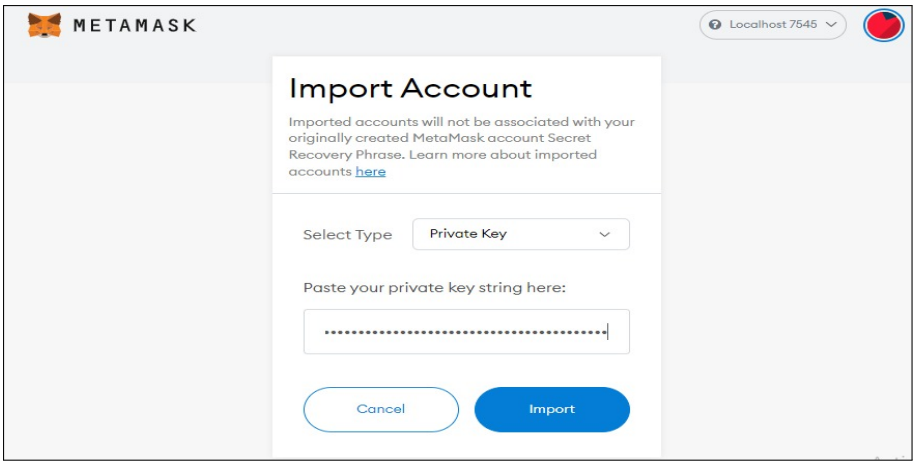

Figure 2.4: Import account by private key in MetaMask

2.2 Running the application

If you successfully installing all components, then you can run the application. Please make sure that you opened Ganache, and connected to MetaMask before running the application.

```
1
2  C:\Users\cemal\OneDrive\Documents\GitHub\Thesis>truffle migrate
3
4  Compiling your contracts...
5  ===========================
6  > Compiling .\contracts\Migrations.sol
7  > Compiling .\contracts\eVote.sol
8  > Artifacts written to C:\Users\cemal\OneDrive\Documents\GitHub\
       Thesis\build\contracts
9  > Compiled successfully using:
10    - solc: 0.5.16+commit.9c3226ce.Emscripten.clang
11
12 Network up to date.
13
14 C:\Users\cemal\OneDrive\Documents\GitHub\Thesis>npm run dev
15
16 > thesis@1.0.0 dev
17 > lite-server
18
19 ** browser-sync config **
20 {
21   injectChanges: false,
```

```
22    files: [ './**/*.{html,htm,css,js}' ],
23    watchOptions: { ignored: 'node_modules' },
24    server: {
25      baseDir: [ './web', './build/contracts' ],
26      middleware: [ [Function (anonymous)], [Function (anonymous)] ]
27    }
28  }
```

Code 2.3: Running the program

As you run the application, you will see the homepage of decentralized e-voting system. Here, you can see "Vote Now" button in the center. If you click on the button, it will direct you to the voting page, but if you haven't logged in, or not registered yet, you won't be able to vote. You need to register, then login to be able to vote, otherwise you won't be able to see the voting page as in Figure 2.6.

Figure 2.5: Homepage of the application

Registration is required for users who don't have an account yet. Users, who already have an account, should click on "Login" and enter their email address and password. Admin login is only used by admins to register candidates, authorize voters, and start/end voting.

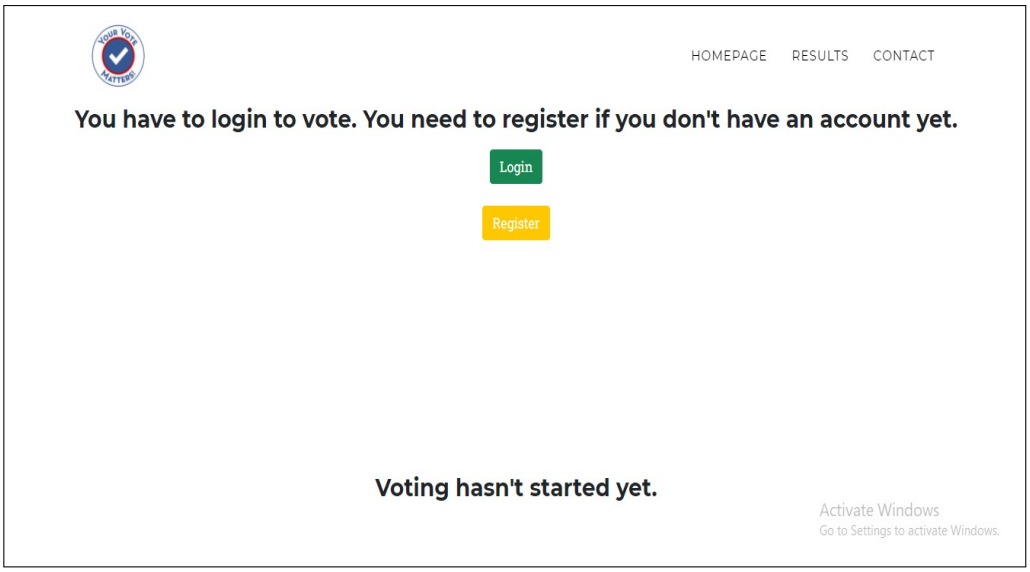

Figure 2.6: Registration/Login is required to be able to vote

When you scroll down on the homepage, you can see the contact section. This section is created to assist voters about their questions, complaints, or requests for improvements.

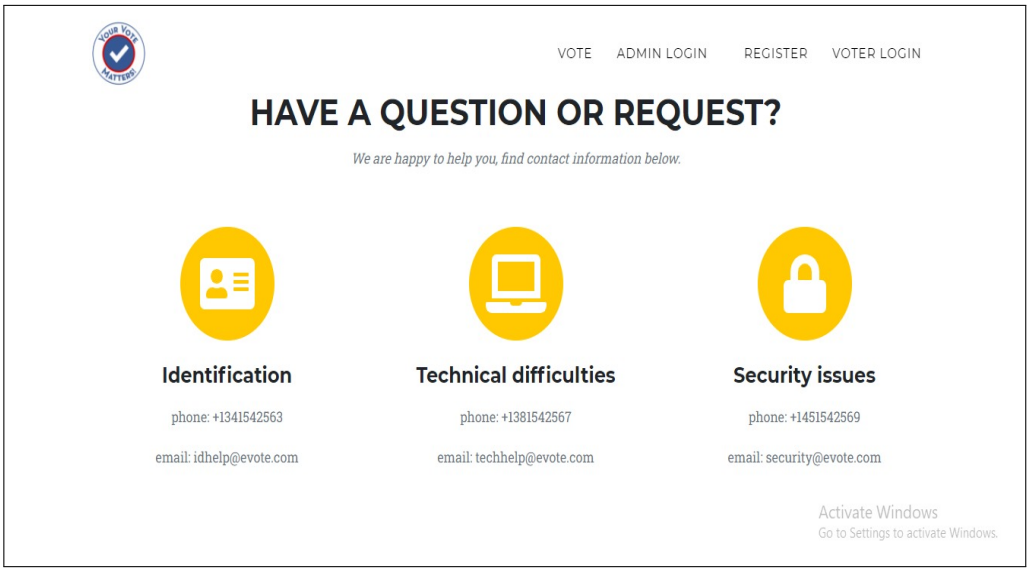

Figure 2.7: Contact page of the application

After user registers, he/she waits for the approval of the admin. Admin will authorize user by their account address, after that user still needs to wait for the admin to start voting process. Authorization of users ensures the eligibility of the voters, that only legitimate electors can cast their vote. When admin starts voting, user will be able to see the candidates and to vote.

When all prerequisites are completed, user is able to vote for the candidate they prefer by clicking on the vote button below the candidate's name. Each user can vote only once. The results will be available when the voting is completed, which is initiated by the admin.

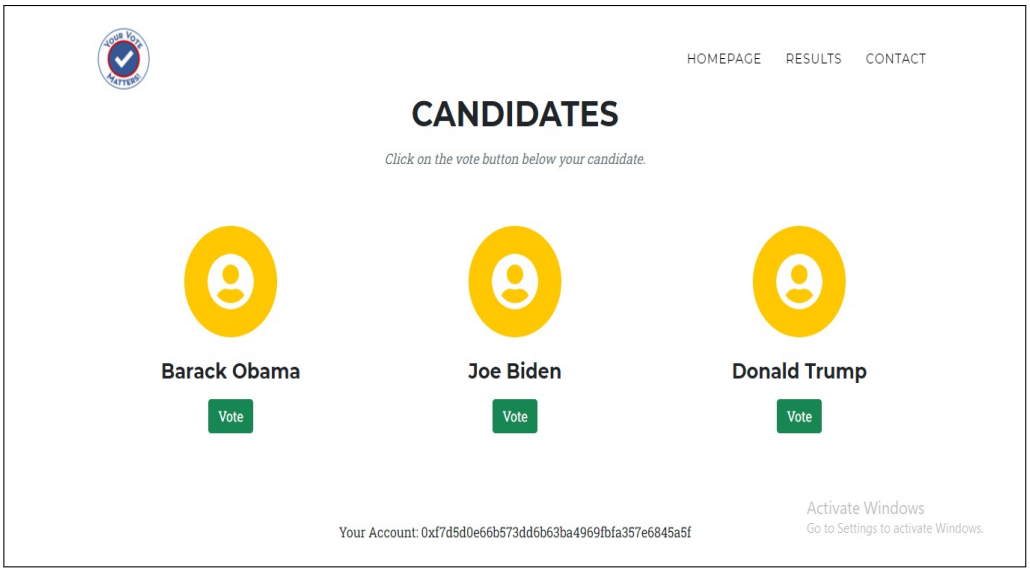

Figure 2.8: Voting page

If two candidates were winner, they gained same amount of votes, then the election will be held again.

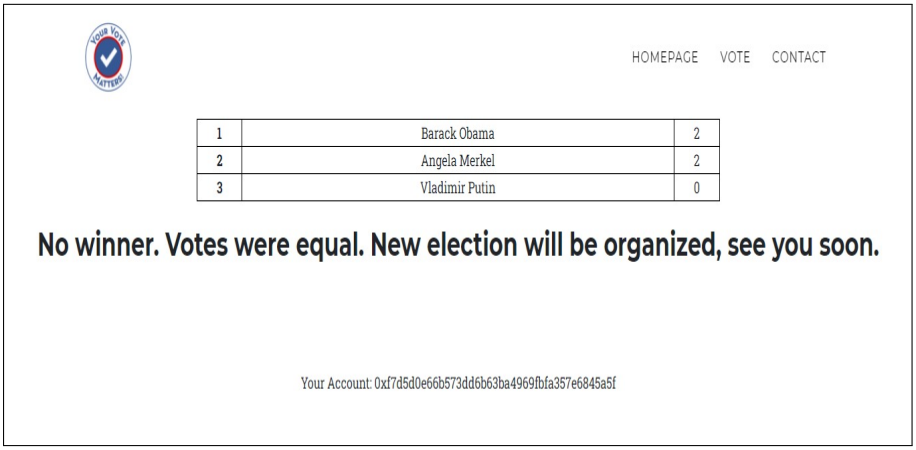

Figure 2.9: Result page of the voting

Chapter 3

Developer documentation

In this chapter, specification, essential parts of the development, and test cases will be covered. Also, use case and diagrams will be provided to demonstrate the workflow of the application. The methods in the smart contract and to interact with smart contract will be explained in detail. Lastly, Ethereum smart contract will be tested thoroughly using truffle framework. This chapter assumes that the developer has basic understanding of smart contracts, web3, Solidity programming knowledge.

3.1 Specification

The voting system intends to provide transparent, decentralized, and convenient electoral process. Following features achieves this:

- Fairness - Smart contract ensures that each voter can only vote once. Voters is able to vote if they are authenticated, and if the voting has started.
- Unhackability - Blockchain features ensure system is secure, only a collusion between 51% of the network would allow for votes to be tampered with. [7]
- Economical - Current voting options costs a lot. Blockchain based voting application reduces the cost significantly.
- Increased speed - Unlike traditional voting tools, online voting speeds up and simplifies the elections.
- Manageability - Administrator authenticates voters' addresses, this helps to maintain security and to reduce fraud.

3.2 Smart Contract Compilation

Developer needs code editor, it can be Notepad++, Visual Studio Code etc, developer may install solidity plugin for the syntax highlighting. Developer should install requirements listed in the User Documentation 2.1 Software Installation Guide.

A dapp consists of two main parts: the contract, which is running in the Ethereum network, and the client. [38] Smart contract is the core of this application, that's why first I'll explain how to compile it first. To compile the smart contract, first developer needs to open the code terminal, and to change directory to the application's path. Then, smart contract can be compiled by using the command in the listing.

```
1
2  C:\Users\cemal\OneDrive\Documents\GitHub\Thesis>truffle compile
3
4  Compiling your contracts...
5  ===========================
6  > Compiling .\contracts\Migrations.sol
7  > Compiling .\contracts\eVote.sol
8  > Artifacts written to C:\Users\cemal\OneDrive\Documents\GitHub\
       Thesis\build\contracts
9  > Compiled successfully using:
10    - solc: 0.5.16+commit.9c3226ce.Emscripten.clang
```

Code 3.1: Smart contract compilation

3.3 Architecture of Decentralized Applications

Architecture of the decentralized applications is illustrated in the diagram below. It can be seen that the client and smart contract is colored in purple, to demonstrate that they're the main parts. Contract is written in the Solidity language and then deployed to the blockchain. Miners run their EVMs (Ethereum Virtual Machines) and handle contract API calls. Such calls can be done using various client libraries. In this application, the client is a web application running in the browser. In order to call Ethereum network API, web3.js library and connection to a Ethereum network node is needed. We may run nodes ourselves or connect to existing one via bridge/proxy e.g. using Metamask. It's a browser plugin, which serves as a bridge for connecting to the network and injecting web3 instance to our code. Metamask

also allows selecting different networks (like a local one for testing) and Ethereum accounts. [38]

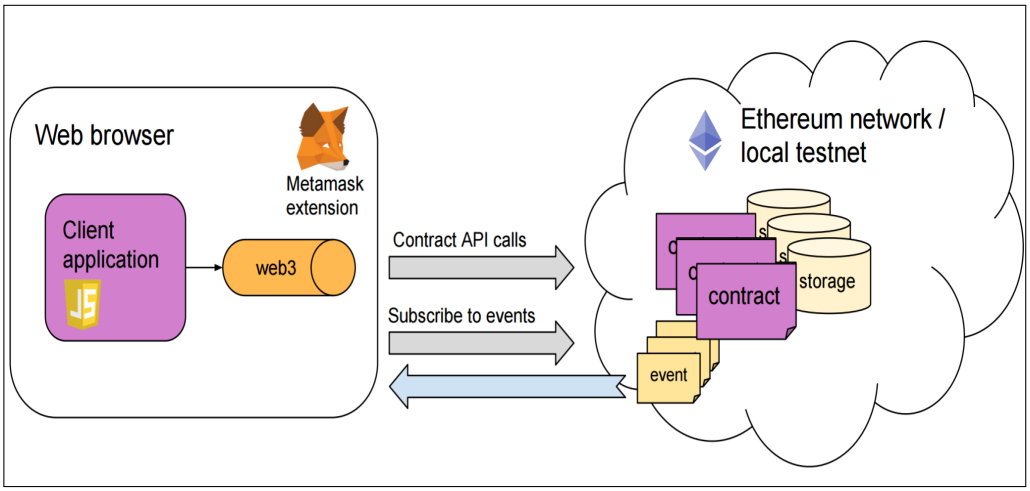

Figure 3.1: Architecture of Decentralized Application [38]

3.4 Development and workflow

The development of the Dapp also proved to be quite a challenging task. The reason for this was the novelty of the technologies used — mainly the Ethereum smart contracts and the Web3.JS library — as well as the lack of correct and updated documentation supporting them.

In the proposed voting system, there are voters and one admin user. Voters can see homepage and contact page of the application without login/registration. After login, they must wait to be authorized by admin user. Authorized users will be able to cast a vote after voting is initiated by admin. For admin user, there are several phases as listed in the use case below. Admin can only take action in the corresponding phase, and so on. There is one more interesting feature for the admin user, he/she is able to monitor voters by viewing the percentage of voted and unvoted users in the pie chart. Use cases for voter and administrator is provided.

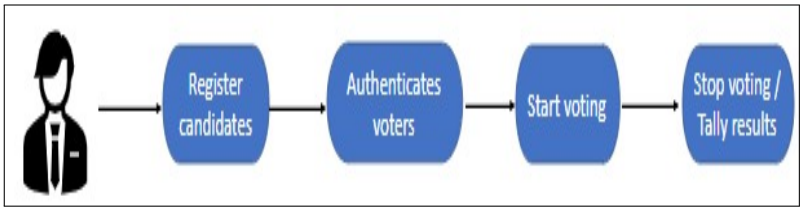

Figure 3.2: Use case for admin

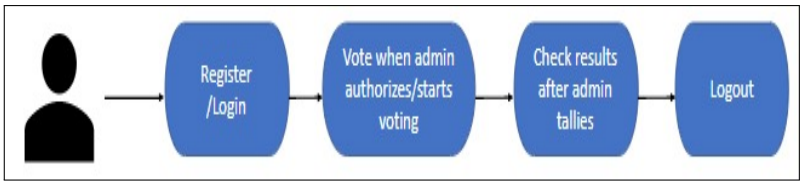

Figure 3.3: Use case for voter

Workflow of the application is demonstrated on the diagram below. Most of the methods requires transaction confirmation from MetaMask, since those methods are contained in the smart contract.

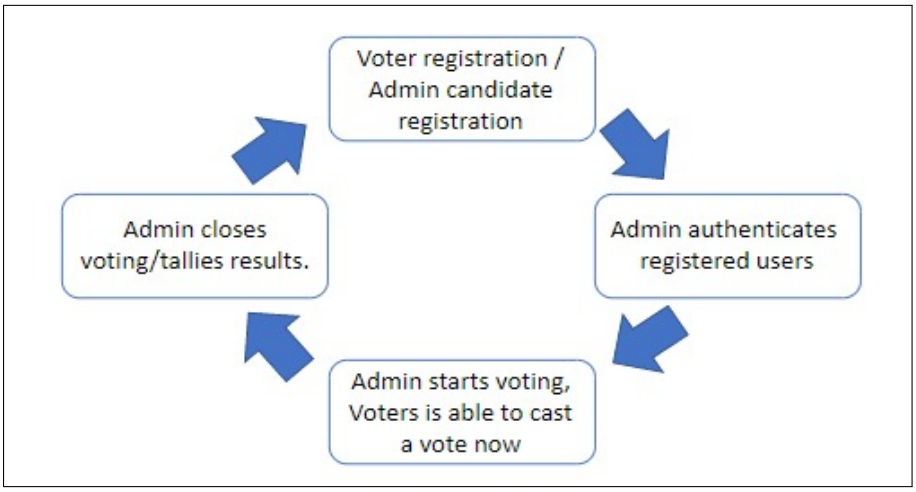

Figure 3.4: Workflow of the application

3.5 Implementation

In this section, I will introduce the structure of voting smart contract and the methods, data types used here. The features for the admin part and the basics for using web3 library to interact with smart contract will be included. The definition of the terms used in the smart contract will be explained.

Smart contract contains structs for voter, candidate, and voter details. Voter has these fields: hasVoted, vote, isRegistered, isLoggedIn, and isAllowedtoVote. Vote field stores candidateID of candidate that the voter has voted. The other fields for the voter is self explanatory boolean variables. VoterDetails struct is used for the registration/login for the users. Candidate struct has id, name, totalVotes fields. Each candidate has an id, it is 1 for the first candidate and increased by one for each new candidate. The field called totalVotes is 0 initially, and it increases when the voters vote for the candidate.

```
struct Voter {
    bool hasVoted;
    uint vote;
  bool isRegistered;
  bool isLoggedIn;
  bool allowedToVote;
}
struct VoterDetails{
  string email;
  string pass;
  string identityno;

}
struct Candidate {
    uint id;
    bytes32 name;
    uint totalVotes;
}
```

Code 3.2: Data types used in smart contract

In Solidity, there is a type called address. Address type stores 20-byte value which is the size of an Ethereum address. In the voting contract, we use this type for the admin user. It can be seen from the code below, that admin user is initialized as "msg.sender". This keyword means the sender of the message, when the user calls smart contract methods, his/her Ethereum address will be "msg.sender". In this case, constructor will be initialized by the Account 1 from Metamask's imported accounts. Hence, the admin account must run the admin methods (adding candidates, authenticating voters, changing phases) from this account. If the admin tries to run these methods from another account, there will be transaction error in the Metamask because smart contract requires those methods to be called by the admin account.

```
address chairman;

constructor() public {
    chairman = msg.sender;
}
```

Code 3.3: Admin is initialized in the constructor

Registering or adding candidate is the first task of admin. To register candidate admin needs to type the name of the candidate in the input field, then through web3.js library addCandidate method of the smart contract will be called.

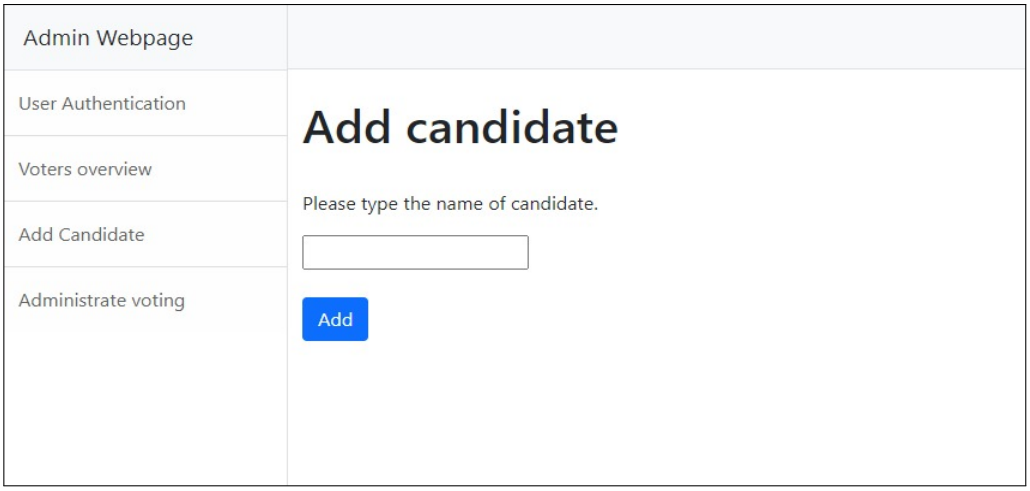

Figure 3.5: Candidate registration

Require keyword checks if the condition is true. If condition is true, then rest of the code will be execute. If it is not correct, then there will be transaction error in the Metamask, that indicates there is exception thrown in contract code. The code checks that only admin can register candidates, and registration is done in the corresponding phase. Id of the candidates starts from 1, and it goes up until the total number of candidates.

```
1  uint public totalCandidates = 0;
2  mapping(uint => Candidate) public candidates;
3
4  function addCandidate(bytes32 _name) public {
5      require(msg.sender == chairman);
6      require(registerCands);
7
8          totalCandidates+= 1;
9          candidates[totalCandidates] = Candidate(totalCandidates,
               _name, 0);
10
11     }
```

Code 3.4: Add Candidate method in the smart contract

Admin changes the phase after the registration of candidates. The following phase is user authentication. In this phase, admin authenticates registered users by their Ethereum address.

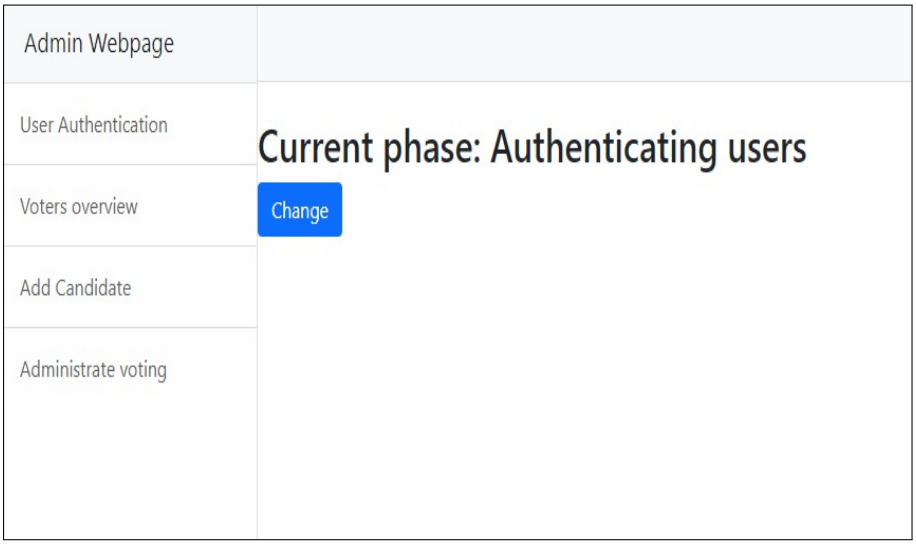

Figure 3.6: User Authentication phase

Then, admin navigates to the user authentication page. Here, all registered users' Ethereum addresses are listed. To authenticate a user, admin should copy the listed address and paste it in the input box. The address is passed as a string to the contract. Html input can't handle 20-byte address, passing it as a number converts hexadecimal address to decimal number. That's why smart contract accept the address as string, then converts string type to address type in the parseAddr function. Admin can only authenticate registered users in the authentication phase.

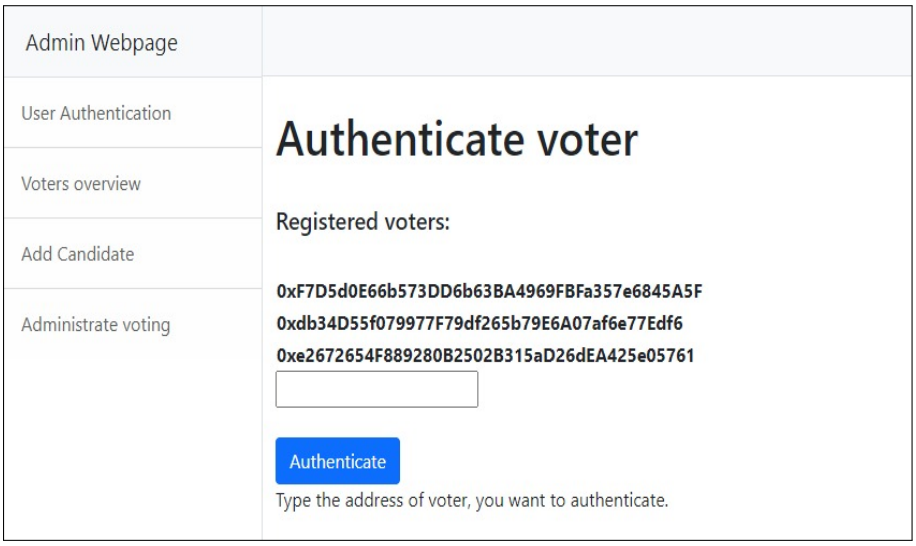

Figure 3.7: Listed users to be authenticated by admin

```
function voterAuth(string memory _voter) public returns (bool) {
  require(msg.sender == chairman);
  require(authorizeUsers);
  address _voter_addr = parseAddr(_voter);
  if(voters[_voter_addr].isRegistered){
    voters[_voter_addr].allowedToVote = true;
  }
  return voters[_voter_addr].allowedToVote;
}
//converts string memory address to solidity type address
function parseAddr(string memory _a) internal pure returns (address
    _parsedAddress) {
    bytes memory tmp = bytes(_a);
    uint160 iaddr = 0;
  uint160 b1;
  uint160 b2;
  for (uint i = 2; i < 2 + 2 * 20; i += 2) {
    iaddr *= 256;
    b1 = uint160(uint8(tmp[i]));
    b2 = uint160(uint8(tmp[i + 1]));
    if ((b1 >= 97) && (b1 <= 102)) {
      b1 -= 87;
    } else if ((b1 >= 65) && (b1 <= 70)) {
      b1 -= 55;
    } else if ((b1 >= 48) && (b1 <= 57)) {
      b1 -= 48;
    }
    if ((b2 >= 97) && (b2 <= 102)) {
      b2 -= 87;
    } else if ((b2 >= 65) && (b2 <= 70)) {
      b2 -= 55;
    } else if ((b2 >= 48) && (b2 <= 57)) {
      b2 -= 48;
    }
    iaddr += (b1 * 16 + b2);
  }
  return address(iaddr);
}
```

Code 3.5: User authentication methods in the smart contract

Voting can be started as the candidates are registered, and registered voters are authenticated by the admin. Admin navigates to administrate voting page, and changes the phase from user authentication to voting started. So, voters are able to cast a vote now. Voters have some prerequisites to cast a vote.

- Voting has started and haven't finished yet.
- Voter hasn't cast a vote yet.
- Voter has been authenticated by admin.
- Voter is voting for the valid candidate (candidate id is in the correct range).

When an event is emitted in Solidity, it stores passed arguments in the transaction log. Events are used to inform the calling application about the current state of the contract, with the help of the logging facility of EVM.

```
1
2  event votedEvent (
3      uint indexed _voteIndex
4  );
5
6  function vote(uint _voteIndex) public {
7    require(startVote && !finishedVote);
8    require(!voters[msg.sender].hasVoted);
9    require(voters[msg.sender].allowedToVote);
10   require(_voteIndex > 0 && _voteIndex <= totalCandidates);
11
12     voters[msg.sender].vote = _voteIndex;
13     voters[msg.sender].hasVoted = true;
14     candidates[_voteIndex].totalVotes += 1;
15     votedUsers += 1;
16     emit votedEvent(_voteIndex);
17 }
```

Code 3.6: Vote method in the smart contract

In the meanwhile, admin is able to view the percentage of the voted and unvoted users. The number of voted and unvoted users are counted in the smart contract. In the jquery file, where we interact with the smart contract, jqplot library is used to plot the pie chart. Jqplot is a plugin for jquery to plot data in different forms of graphs and charts. This chart is quite useful for the voting application, since it can be decided whether the election had active participation or not.

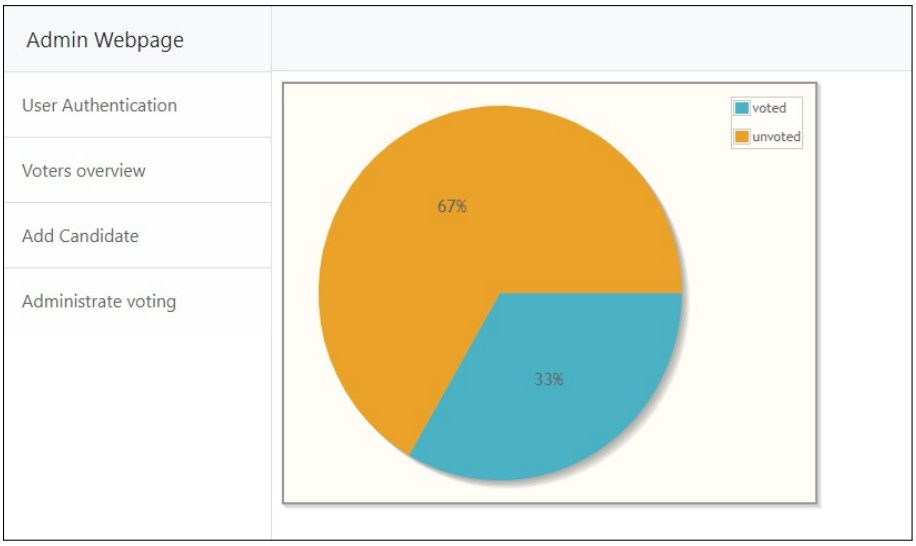

Figure 3.8: Pie chart for voters overview

The last task of the admin is to stop voting. When admin stops voting, admin and all the voters will be able to see the results.

Figure 3.9: Election results after admin ends voting

3.6 Testing

Testing smart contracts is quite important. If there is a problem in the smart contract, it may lead to serious damages and costs afterwards. For this application, smart contract is tested by using truffle framework. Testing is the most important aspect of quality smart contract development. Truffle tests will be unit test, since this only tests if smart contract methods working correctly in various cases.

Truffle contains automated testing framework to make testing contracts easier. All test files should be located in the ./test directory. Truffle will only run test files with the following file extensions: ".js, .ts, .es, .es6, and .jsx, and .sol". All other files

are ignored. When "truffle test" command is executed in terminal, truffle checks the folder and runs the tests.

```
1
2  C:\Users\cemal\OneDrive\Documents\GitHub\Thesis>truffle test
3  Using network 'development'.
4
5
6  Compiling your contracts...
7  ===========================
8  > Compiling .\contracts\Migrations.sol
9  > Compiling .\contracts\eVote.sol
10 > Compiling .\contracts\eVote.sol
11 > Artifacts written to C:\Users\cemal\AppData\Local\Temp\test
       --7576-wxiSgEtztvRY
12 > Compiled successfully using:
13    - solc: 0.5.16+commit.9c3226ce.Emscripten.clang
```

Code 3.7: Truffle run command for unit tests

Unit tests are good to small code pieces such as methods, functions, classes. Developer checks the speed of code, and the output for different test cases. I tested methods in the smart contract for unit testing using truffle test framework. Truffle uses the Mocha testing framework and Chai assertion library to provide a solid framework for JavaScript test. JavaScript test gives you more flexibility and enables you to write more complex tests.

```
1 var eVote = artifacts.require("./eVote.sol");
```

Code 3.8: Writing unit test for smart contract

Truffle saves your artifacts when you compile and updates them with address information when you migrate. Pretty much whenever you use any part of Truffle, these artifacts are read from disk and processed, enabling Truffle to paint itself a picture of what's happening in human terms. Artifacts.require() method is used to request a usable contract abstraction for a specific Solidity contract.[39]

```
1 contract("eVote", function(accounts) {
2   var evoteInstance;
3
4   it("starts with 0 candidates", function() {
5     return eVote.deployed().then(function(instance) {
```

```
6     evoteInstance = instance;
7       return evoteInstance.totalCandidates();
8     }).then(function(candidates) {
9     assert.equal(candidates, 0);
10  });
11  });
12
13  it("admin adds first candidate successfully", function() {
14    return eVote.deployed().then(function(instance) {
15    evoteInstance = instance;
16      return evoteInstance.addCandidate(web3.utils.fromAscii("
          Angela"), { from: accounts[0] })
17    }).then(function(receipt) {
18    return evoteInstance.totalCandidates();
19    }).then(function(candidates) {
20    assert.equal(candidates, 1);
21  });
22  });
23
24 });
```

Code 3.9: Initial unit tests for smart contract

The contract() provides a list of account available by the Ganache which can be used to write tests. Furthermore, it also groups together all the tests for a specific contract. "it()" is a Mocha notation for a test case. The it call identifies each individual tests but by itself it does not tell Mocha anything about how your test suite is structured. You can see that we have created the instance of smart contract to access its methods. Then we are asserting the output of the methods to see if it works correctly.

```
1   it("admin can't validate unregistered user", function() {
2     return eVote.deployed().then(function(instance) {
3     evoteInstance = instance;
4       return evoteInstance.voterAuth(accounts[2],{ from: accounts
          [0]})
5     }).then(function(receipt) {
6     return evoteInstance.voters(accounts[2]);
7     }).then(function(voter) {
8     assert.equal(false, voter[2]);
9   });
```

```
10  });
11
12  it("throws an exception for invalid candidates", function() {
13    return eVote.deployed().then(function(instance) {
14    evoteInstance = instance;
15      return evoteInstance.vote(44,{from: accounts[0]})
16    }).then(assert.fail).catch(function(error) {
17      assert(error.toString().indexOf('revert') >= 0, "error
          message must contain revert");
18    });
19  });
20
21  it("throws an exception for double voting", function() {
22    return eVote.deployed().then(function(instance) {
23    evoteInstance = instance;
24      return evoteInstance.vote(1,{from: accounts[0]});
25    }).then(assert.fail).catch(function(error) {
26      assert(error.toString().indexOf('revert') >= 0, "error
          message must contain revert");
27    });
28  });
```

Code 3.10: Adverse test scenarios for unit testing

Negative test cases are performed to try to "break" the software by performing invalid (or unacceptable) actions, or by using invalid data. It tests stability of the application. The application's functional reliability can be quantified only with effectively designed negative scenarios. You may see some negative testing scenarios for the voting smart contract. In the first test case, admin tries to authenticate unregistered user. Assertion ensures the user is not authenticated by checking the voter field is false. Second test case throws an exception, because voter tries to vote for a candidate with id 44, which doesn't exists. Voter can't vote more than once, smart contract doesn't allow this to happen, third case simulates this, and smart contract throws an exception.

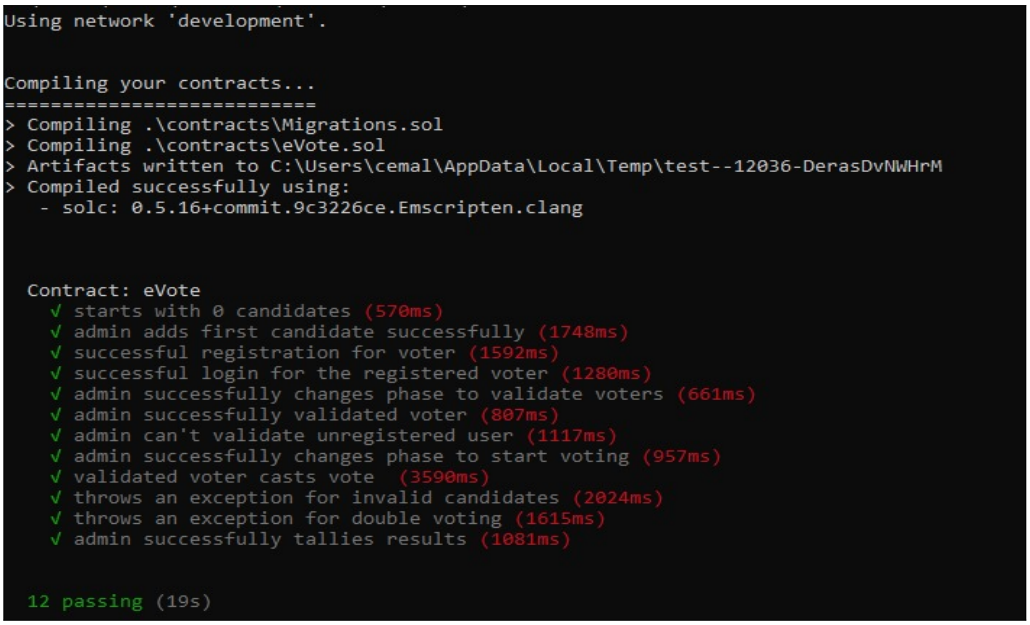

Figure 3.10: The result of unit tests for the smart contract

There are 12 unit test cases so far, but the number of tests can be increased. Test cases include positive and negative testing. Positive testing based on correct actions to prove that application works as desired, negative testing, on the contrary, checks if invalid actions can cause any problems/errors. Through the provided test cases, we can make sure the smart contract works correctly.

Chapter 4

Conclusion and Future Work

This thesis work presented that decentralized voting system provides secure and transparent voting experience for voters, makes a difference compared to other online voting applications with its highly secure underlying blockchain technology. Blockchain technology's distributed ledger technology doesn't allow to delete or revert previous transactions, thus the security increases since database can be seen by everyone and system can't be manipulated.

The voting security is enhanced by providing admin functionality. Admin is the one who adds candidates, and authenticates voters. In the proposed system, authentication can be done using voter address. It is supposed that the admin knows which user addresses should be authenticated. It is also possible to view the percentage of voted and unvoted users in real time. Unlike traditional voting systems, operations are fast and reliable because of reduced human errors.

In this application, I utilized smart contracts which makes blockchain programmable. Smart contract forms trust between voter and application, eliminating the need for central authority. The proposed application can be used for small-scaled elections. Security enhancements are must be taken to use blockchain based voting applications on larger scale, because the blockchain technology itself is still evolving and changing.

This application can be improved further by changing user registration and login to more secure process using biometrics. Mobile application can be developed for this web application, this can help to reach greater audience.

Bibliography

[1] Nangula Shejavali. "Electronic Voting Machines". In: *Institute for Public Policy Research (IPPR) No* 1 (2014).

[2] Jeannette Lynn Fraser. "THE EFFECTS OF VOTING SYSTEMS ON VOTER PARTICIPATION: PUNCH CARD VOTING SYSTEMS IN OHIO (MACHINES, ELECTION ADMINISTRATION, OVERVOTING, EQUIPMENT, BALLOT FORM)". PhD thesis. The Ohio State University, 1985.

[3] Andrew W Appel, Richard A DeMillo, and Philip B Stark. "Ballot-marking devices cannot ensure the will of the voters". In: *Election Law Journal: Rules, Politics, and Policy* 19.3 (2020), pp. 432–450.

[4] Baocheng Wang et al. "Large-scale Election Based On Blockchain". In: *Procedia Computer Science* 129 (2018), pp. 234–237. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.03.063. URL: https://www.sciencedirect.com/science/article/pii/S1877050918302874.

[5] Lejun Zhang et al. "Secure and efficient data storage and sharing scheme for blockchain-based mobile-edge computing". In: *Transactions on Emerging Telecommunications Technologies* (2021), e4315.

[6] Ori Jacobovitz. "Blockchain for identity management". In: *The Lynne and William Frankel Center for Computer Science Department of Computer Science. Ben-Gurion University, Beer Sheva* (2016).

[7] Jennifer J Xu. "Are blockchains immune to all malicious attacks?" In: *Financial Innovation* 2.1 (2016), pp. 1–9.

[8] Jae Hyung Lee et al. "Systematic approach to analyzing security and vulnerabilities of blockchain systems". PhD thesis. Massachusetts Institute of Technology, 2019.

[9] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". In: *Decentralized Business Review* (2008), p. 21260.

[10] *What is a block in the blockchain.* https://medium.datadriveninvestor.com/what-is-a-block-in-the-blockchain-c7a420270373/.

[11] *Blockchain Structure.* https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch07.html/.

[12] *Merkle Root (Cryptocurrency).* https://www.investopedia.com/terms/m/merkle-root-cryptocurrency.asp/.

[13] *What is Merkle Tree?* https://cointral.com/what-is-merkle-tree/.

[14] *Blockchain - Quick Guide.* https://www.tutorialspoint.com/blockchain/blockchain_quick_guide.htm/.

[15] *PoS explained.* https://academy.binance.com/en/articles/proof-of-stake-explained/.

[16] *Let's learn more about the Proof of Stake.* https://www.reddit.com/user/btcbamofficial/comments/oleo79/lets_learn_more_about_the_proof_of_stake/.

[17] *SHA-256.* https://en.bitcoinwiki.org/wiki/SHA-256/.

[18] *SHA-256 Algorithm.* https://komodoplatform.com/en/academy/sha-256-algorithm/.

[19] Simanta Sarmah. "Understanding Blockchain Technology". In: 8 (Aug. 2018), pp. 23–29. DOI: 10.5923/j.computer.20180802.02.

[20] *Different types of blockchain technologies.* https://thebossmagazine.com/different-types-of-blockchain-technologies/.

[21] *Blockchain usage in healthcare.* https://guardtime.com/health/.

[22] *Blockchain usage in ATLANTA project.* https://www.newsbtc.com/news/atlant-blockchain-real-estate-ecosystem/.

[23] *Supply chain management.* https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7522652/.

[24] *What is Double Spending.* https://corporatefinanceinstitute.com/resources/knowledge/other/double-spending/.

[25] *Bitcoin logo*. https://logos-world.net/bitcoin-logo/.

[26] *What happens bitcoin after 21 million mined*. https://www.investopedia.com/tech/what-happens-bitcoin-after-21-million-mined/.

[27] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[28] *Ethereum docs - Account Types, Gas, and Transactions*. https://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html/.

[29] Shuai Wang et al. "Blockchain-enabled smart contracts: architecture, applications, and future trends". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49.11 (2019), pp. 2266–2277.

[30] *What is Smart Contract*. https://developers.rsk.co/guides/full-stack-dapp-on-rsk/part1-overview/.

[31] *Introduction to Solidity*. https://www.geeksforgeeks.org/introduction-to-solidity/.

[32] *NodeJS installation v14.17.5*. https://nodejs.org/ko/blog/release/v14.17.5/.

[33] *Features Of Truffle Ethereum*. https://www.edureka.co/blog/developing-ethereum-dapps-with-truffle/.

[34] *What is Truffle Suite*. https://www.upgrad.com/blog/what-is-truffle-suite/.

[35] *Install Ganache*. https://www.trufflesuite.com/ganache/.

[36] *Web2 vs Web3*. https://ethereum.org/en/developers/docs/web2-vs-web3/.

[37] *MetaMask Chrome*. https://chrome.google.com/webstore/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn?hl=en/.

[38] *Architecture of a dapp*. https://softwaremill.com/event-sourcing-on-blockchain/.

[39] *Architecture of a dapp*. https://trufflesuite.com/blog/introducing-truffle-db-part-1/.

List of Figures

List of Codes