

CSCI-652 TERM PROJECT REPORT

Topic: Blockchain Based Voting System using Smart Contracts

Authors: 1. Divith Reddy Gajjala <dg6110@g.rit.edu>
2. Chirayu Anil Marathe <cm6647@g.rit.edu>
3. Shweta Sandip Sharma <ss6505@g.rit.edu>

1. Introduction

In traditional voting applications which are based on a centralized approach, there is a general convention that the applications are usually provided by a centralized database handled by an administrator. When voting via a centralized application one big concern is security. The results may be tampered with or there could be attacks on the system to modify the results to a desired outcome. Another issue with centralized systems is that each server typically provides only a single function thereby making it a single point of failure. This means that in the event of an attack on a specific node, the entire system will fail.

To overcome these obstacles, we use a **decentralized** voting application. It is based on two principles- a **peer to peer network** is incorporated with the application logic and executed independently on every server. Here, a central node doesn't co-ordinate the servers. The servers communicate with each other directly. The second principle is called **blockchain guarantees**. This implies that the data cannot be modified in a retroactive manner. A decentralized voting application can make security breaches futile by replicating the execution over a network which includes several servers.

A blockchain database is used which is based on a chain of blocks where a block is a record holding a set of digitally signed transactions, some metadata and a link to the previous block. A blockchain database addresses security concerns by protecting against attacks from malicious parties.

2. Outline

The blockchain acts as a network and a database all in one. It is a peer-to-peer network of computers which are referred to as individual nodes. These nodes share the data and code in the network. The most important aspect of the blockchain is that **there are no central servers**. All of the data shared across the nodes in the blockchain is contained in blocks which are bundles of records chained together to form a **public ledger**. This represents all of the data in the blockchain. The reason we are using blockchain for our voting application is that it will help us ensure that our vote was counted and did not change. This is achieved seamlessly since the blockchain incorporates **cryptographic hash functions** to enable security and leverages **consensus algorithms** for validation.

Decentralized Application (Dapp)- It is an application which includes a description of components of the client side and the server side. It is a peer-to-peer application where the data and code are shared.

Ethereum Network- The participating nodes host a blockchain database along with a node client which facilitates a node to communicate with all the other nodes in the system. Within the Ethereum network there is no clear concept of client and server since all the nodes are equivalent to each other.

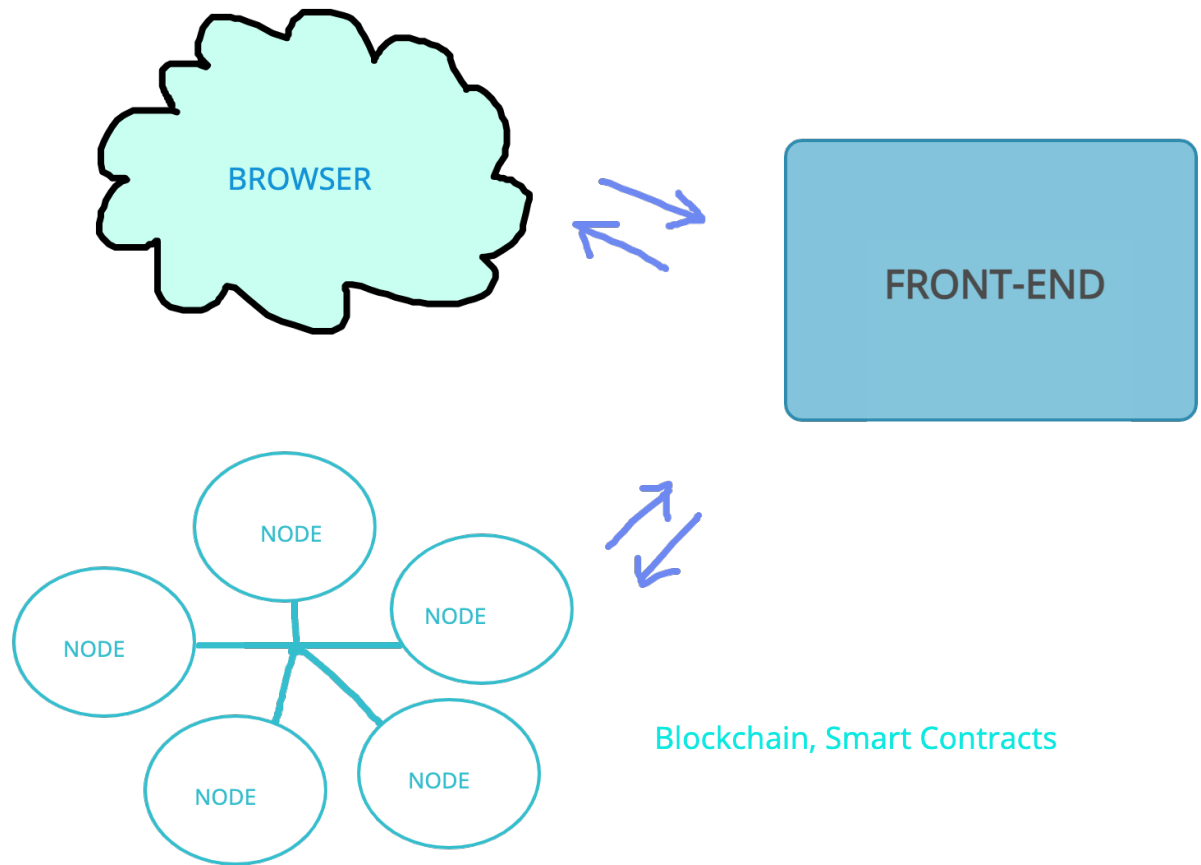
Ethereum clients can expose a common client interface and enable communication with each other via a P2P protocol- **Wire**. This protocol enforces a standard to send data through the network especially a transaction- in this case a submitted vote and a block (set of votes in the blockchain database).

The code for decentralize applications is structured into smart contracts which can encapsulate the state and logic. The decentralized voting application would be structured on several kinds of smart contracts that can be hosted on the Ethereum blockchain.

Network Nodes- Some nodes have nodes have different functions, here are two main kinds of nodes as follows:

1. **Full node**- These nodes propagate the votes received from their peers to other peers in the network. They verify whether the blocks received are correct and contain authentic votes by running the voting Dapp smart contracts. However, full nodes do not store votes in the new blockchain blocks.
2. **Mining node**- These nodes are configured to perform processing of transactions in an active manner. They can group and store the transactions processed in new blocks. They are rewarded in Ether, the cryptocurrency of Ethereum platform. These blocks are then propagated to the rest of the P2P network. These nodes are called mining nodes since the process of consolidating a new block to that blockchain and receiving the reward for it is known as mining. In this project, the mining nodes group the votes received from the peer nodes into a new block and append this block to the blockchain.

Smart Contract- Similar to a micro-service that lives on the web. This is where all the **business logic**(coding) of the project lies. This is in charge of **reading/writing data, transferring value and executing any business logic that we code**. It holds an agreement that each vote in the blockchain will be counted only once, won't change and the correct candidate will always win. This is written in the programming language **Solidity**.



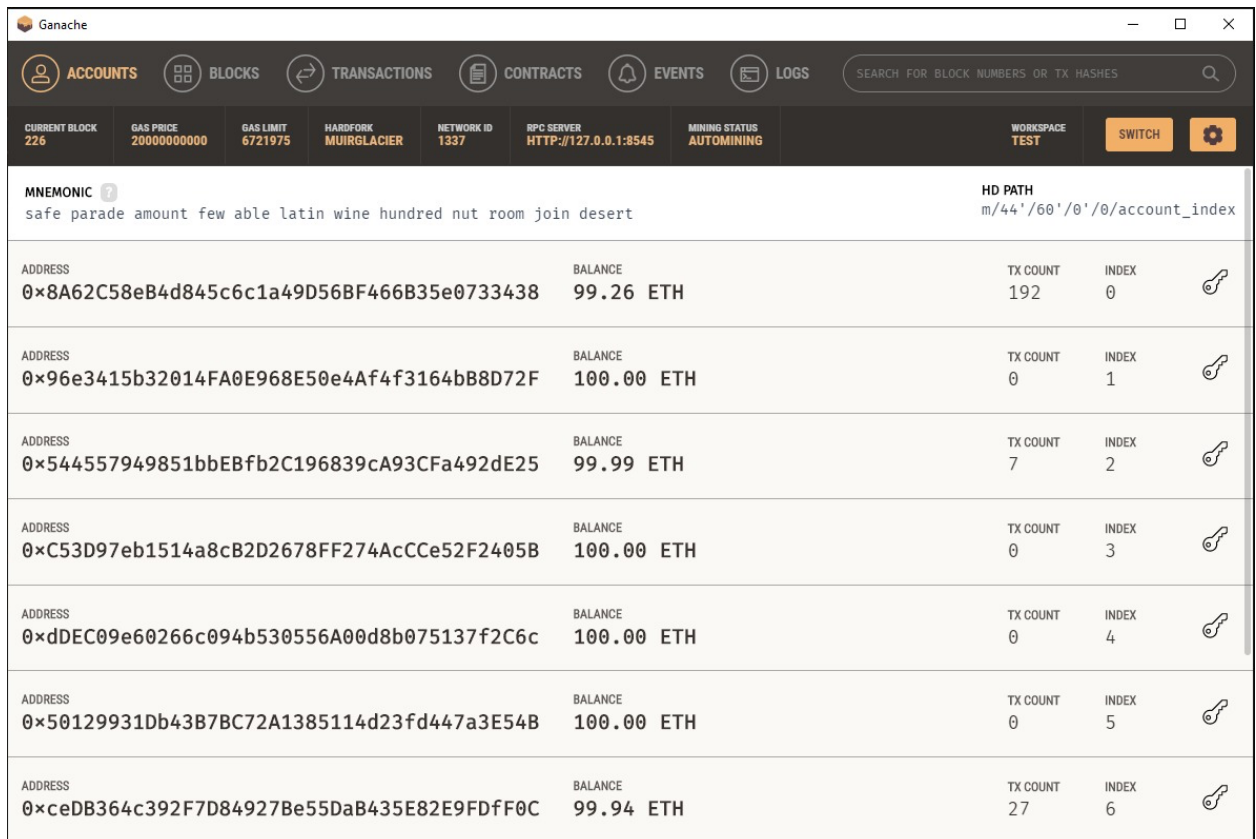
High level overview of the application

Dependencies needed:

Node Package Manager (NPM)- This is the first dependency needed which comes with Node.js.

Truffle- A framework that allows us to create centralized applications on the Ethereum network. Gives us a suite of tools that allow us to write smart contracts on Solidity. It allows provides a framework for testing our smart contracts and deploying them on the blockchain. We will be using the Pet Shop Box here, which includes the basic project structure along with the code.

Ganache- A local end-memory blockchain. Starts running a local blockchain on the machine. In the project, it starts with ten accounts. Each account represents the voters in the election and have a unique address. The accounts have been credited with 100.00 ETH.



The screenshot shows the Ganache application window. At the top, there's a navigation bar with icons for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below this is a status bar displaying various network metrics: CURRENT BLOCK (226), GAS PRICE (2000000000), GAS LIMIT (6721975), HARDFORK (MUIRGLACIER), NETWORK ID (1337), RPC SERVER (HTTP://127.0.0.1:8545), MINING STATUS (AUTOMINING), and WORKSPACE (TEST). A search bar is also present. The main area displays the MNEMONIC (safe parade amount few able latin wine hundred nut room join desert) and the HD PATH (m/44'/60'/0'/0/account_index). Below this is a table of accounts.

ADDRESS	BALANCE	TX COUNT	INDEX	
0x8A62C58eB4d845c6c1a49D56BF466B35e0733438	99.26 ETH	192	0	
0x96e3415b32014FA0E968E50e4Af4f3164bB8D72F	100.00 ETH	0	1	
0x544557949851bbEBfb2C196839cA93CFa492dE25	99.99 ETH	7	2	
0xC53D97eb1514a8cB2D2678FF274AcCCe52F2405B	100.00 ETH	0	3	
0xdDEC09e60266c094b530556A00d8b075137f2C6c	100.00 ETH	0	4	
0x50129931Db43B7BC72A1385114d23fd447a3E54B	100.00 ETH	0	5	
0xceDB364c392F7D84927Be55DaB435E82E9FDfF0C	99.94 ETH	27	6	

The image above depicts the interface of Ganache. There is a track on the current block, gas price, gas limit, hardfork (a radical change in the network's protocol that makes the previously invalid transactions or blocks valid), network ID, RPC server and mining status. Each candidate has its own unique address and balance in ETH.

MetaMask- A Google Chrome extension that allows to connect to the local Ethereum network and interact with the smart contracts. Keeps track of all the candidate accounts by providing authentication to them and the administrator.

MetaMask connects with Ganache- once the blockchain node is connected, an account needs to be added in the MetaMask wallet. When MetaMask is added, it exposes the object globally. If available, it implies that MetaMask is enabled. Any operation performed at this point will be using the connected wallet.

We have to switch accounts between the admin and other non-admin users with the Metamask extension. Each account on MetaMask represents a node on the blockchain. This way we can work with different nodes of the network using a single machine.

A unique wallet address is also set by Ganache for each account since every time a value is passed in the blockchain, there is actually a transaction being made thus, the gas amount would be deducted from the designated wallet.

One of the roadblocks encountered while building this project was authentication via MetaMask. As the Metamask extension is deprecated, several commands are not supported

during the time this project was deployed. The use of this extension caused a few inexplicable issues with the code. Additionally, the extension also caused Google Chrome to crash several times without any warning or explanation. There was also difficulty in creating a clean workaround and the best outcome was to resort to a patchwork solution in certain areas. For instance, while rendering the Candidate Voting window and the Election Results window, the MetaMask extension does not work if each element in the page is rendered only once. The only solution to overcome this roadblock is to get the extension to work in sync with the webpage by rendering every element twice.

3. Results

An electronic voting system running on the blockchain will significantly decrease the cost of running an election and help improve voter participation, all while preserving the integrity of the election in a tamper-proof manner.

As a result, new users are able to register and get approved by the admin. After that, they are re-directed to a page where they can cast their vote. The vote can only be cast once per candidate and can not be changed once placed. Thus, the achievement of this project is that the votes are stored in a tamper-proof manner due to the blockchain in the back end. All of this takes place in a decentralized manner and is highly secure due to blockchains' use of cryptographic hash functions and the Proof of Work (PoW) consensus model.

Election Results		
#	Name	Votes
1	Divith	1
1	Divith	1
2	Harry	1
2	Harry	1

Your Account: 0x96e3415b32014fa0e968e50e4af4f3164bb8d72f

Final Election Results after voting concludes

4. Documented Code

The GitHub link for all the code in JavaScript and Solidity is provided here-

https://github.com/Cam-98/DSTermProject_Election

There is also a demo video which demonstrates how the application runs and the results it achieves.

5. Manual

The project has a straight-forward and simplistic approach. The sequence of steps is mentioned below:

1. First, we create an empty directory and clone the repository containing the project files using the command – `git clone https://github.com/Cam-98/DSTermProject_Election.git`
2. Then, we compile the Solidity contracts which actually implement the voting system on the blockchain using the command – `truffle migrate --reset`

```
C:\Users\chira\Desktop\Semester 3 Fall 2021\CSCI 652- Distributed Systems\Term Project\Application Code\test>truffle migrate --reset

Compiling your contracts...
=====
> Compiling .\contracts\Election.sol
> Compiling .\contracts\Election.sol
> Compiling .\contracts\Migrations.sol
> Artifacts written to C:\Users\chira\Desktop\Semester 3 Fall 2021\CSCI 652- Distributed Systems\Term Project\Application Code\test\build\contracts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang

> Something went wrong while attempting to connect to the network at http://127.0.0.1:8545. Check your network configuration.

Could not connect to your Ethereum client with the following parameters:
  - host      > 127.0.0.1
  - port      > 8545
  - network_id > *
Please check that your Ethereum client:
  - is running
  - is accepting RPC connections (i.e., "--rpc" or "--http" option is used in geth)
  - is accessible over the network
  - is properly configured in your Truffle configuration file (truffle-config.js)

Truffle v5.4.23 (core: 5.4.23)
Node v16.13.1
```

Compiling the contracts

3. Next, we start the server containing the web application using the command- `npm run dev`

Once the server starts, we are redirected to the web page containing the application.

```
C:\Users\chira\Desktop\Semester 3 Fall 2021\CSCI 652- Distributed Systems\Term Project\Application Code\test>npm run dev

> pat-shop@1.0.0 dev
> lite-server

** browser-sync config **
{
  injectChanges: false,
  files: [ './**/*.{html,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://192.168.0.1:3000
  UI: http://localhost:3001
  UI External: http://localhost:3001

[Browsersync] Serving files from: ./src
[Browsersync] Serving files from: ./build/contracts
[Browsersync] Matching files...
21.12.10 20:40:16 304 GET /index.html
21.12.10 20:40:16 304 GET /css/bootstrap.min.css
21.12.10 20:40:16 304 GET /js/bootstrap.min.js
21.12.10 20:40:16 304 GET /js/web3.min.js
21.12.10 20:40:16 304 GET /js/truffle-contract.js
21.12.10 20:40:16 304 GET /js/app.js
(node:6768) [DEP0066] DeprecationWarning: OutgoingMessage.prototype._headers is deprecated
(Use "node --trace-deprecation ..." to show where the warning was created)
21.12.10 20:40:17 200 GET /election.json
```

Running the server

4. The current implementation of the system has only one single admin. Every other user first must register his/her details with the system.

Election Results

Register

Please fill in this form to create an account.

Name

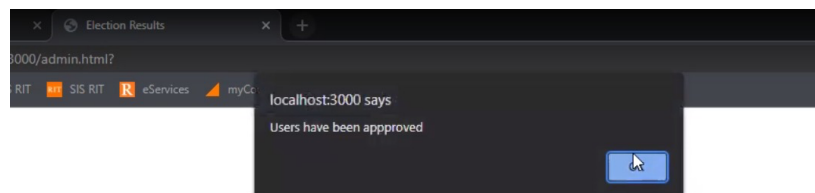
Email

ADMIN

Register

Registration Window for new users

- Once the user enters his details as depicted in step 4, their entry will be saved to the admin's page. The admin first approves users among those who have registered. Then the admin selects candidates among all the approved users.



NEW USERS

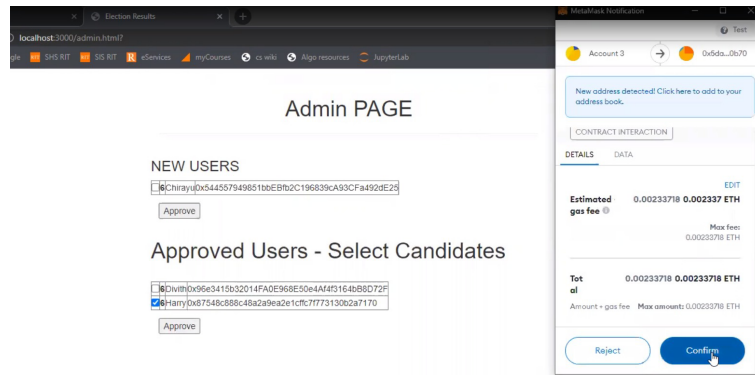
<input checked="" type="checkbox"/>	Divith	0x96e3415b32014FA0E968E50e4Af4f3164bB8D72F
<input checked="" type="checkbox"/>	Harry	0x87548c888c48a2a9ea2e1cfc7f773130b2a7170
<input type="checkbox"/>	Chirayu	0x544557949851bbEBfb2C196839cA93CFa492dE25

Approve

Admin approving registered users (Admin view of the system)

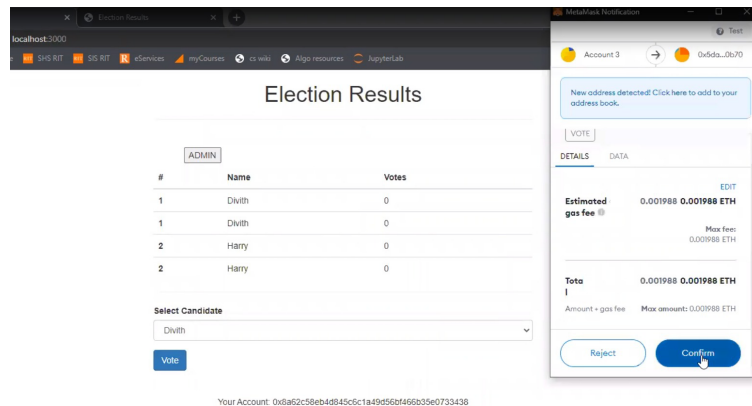
- Each operation is charged a small amount of fee (in ETH) and is recorded on the blockchain. This ensures that a log of all operations is maintained. As each operation records the account details of all entities involved in the transaction, the legitimacy of the transaction can be verified easily. This is done using the MetaMask Chrome web

extension which connects the accounts running on the local blockchain created using Ganache to the users registering with the server.



Admin selecting Candidates from approved users

7. The approved users and candidates can now cast their vote which will be stored in the system. Each vote is recorded on the blockchain and all the vote counts are updated automatically. Once a vote is cast, it cannot be manipulated or tampered with. Any other candidates attempting to vote will be able to view the status of vote count of each candidate.



Candidate casting a vote for a candidate of their choice

8. Once a user has cast a vote, he/she will be redirected to the Election results window. Here the results will be displayed and updated in real time as each user casts a vote. The results of the votes will indicate the total number of votes secured by each candidate in the ballot.

Election Results

#	Name	Votes
1	Divith	0
1	Divith	0
2	Harry	0
2	Harry	0

Select Candidate

Divith

Your Account: 0x8a62c58eb4d845c5c1a49d5ebf466b35e0733438

Election Results window each candidate see after casting a vote

6. Conclusions

The project successfully implements a blockchain-based voting system.

Future Scope:

1. Discovering a suitable replacement for the MetaMask chrome extension which was responsible for the double rendering discussed earlier.
2. There could be a feature to make the admin selection process dynamic instead of hard coding a value for the admin account. This would enable users to appoint themselves as admins and effectively create their own elections.
3. To improve convenience and ease of access, this project could be implemented via mobile devices.

The reason behind using blockchain behind the scenes is because it is very reliable and tamper-proof. The way security is incorporated into the blockchain is because of its' use of cryptographic hash functions. Additionally, it leverages a very popular consensus model- Proof of Work (PoW) which is a decentralized mechanism which requires the members of the network to participate in expanded effort solving in an arbitrary mathematical puzzle. With the advent of PoW, there is assurance that the data stored in one block cannot be tampered with without re-calculating the PoW for each subsequent block. Thus, with the blockchain, the need for a trusted third party can be completely eradicated. All of this takes place behind the scenes and is the crux of this project which emphasizes on a decentralized blockchain-based system for high security.

7. References

1. <https://livebook.manning.com/book/building-ethereum-dapps/chapter-1/69>
2. <https://www.mdpi.com/1424-8220/21/17/5874/pdf>
3. https://www.researchgate.net/publication/351723225_Blockchain_Based_E-Voting_System