# Decentralized Election System

**Project & Thesis-I CSE 4100**

Submitted by

| | |
|---|---|
| **Monjure Mowla** | **180104027** |
| **Ameer Talha** | **180104036** |
| **Nipun Paul** | **180104042** |
| **Mutakabbirul Islam Pranto** | **180104044** |

Supervised by

**Mr. Md. Khairul Hasan**

**Department of Computer Science and Engineering**

**Ahsanullah University of Science and Technology**

Dhaka, Bangladesh

June 26, 2022

# ABSTRACT

Blockchain as a new technology provides innumerable opportunities. Some very critical issues that are existing in our current society, can be solved with the help of blockchain. Blockchain is an immutable distributed ledger and these properties can be used to solve a very sensitive problem such as  a fair election. In this paper, we have created a decentralized election system using the ethereum blockchain. On our project, the voters can easily vote for their candidate of choice and feel assured that their votes will not be manipulated. Because, since the smart contract of this project is deployed on the ethereum blockchain, nobody can tamper with the voting. We also made sure that nobody can vote multiple times. We have the target to improve the e-government system. As such, we have taken the initiative to solve this problem of fraudulent elections and therefore, make the job of the Election Commission and the Government easier.

# Contents

# List of Figures

# Chapter 1

# Introduction

E-government services are essential in the modern era. To make the existing services of the government easily available, e-government is a necessary step. Perhaps the most crucial element in the government system is the fair election to determine the popular candidate or choice. Thus, a digital election can be the most vital example of e-government system. But holding a fair election digitally will be very challenging. This is where blockchain comes in. The three most important properties of this technology is: (1) Decentralized, (2) Immutability and (3) Transparency. These characteristics of blockchain make the e-voting system into a reality.

# Chapter 2

# Background Study

In order to build our project, we have learnt the following concepts:

## 2.1 Blockchain

With the development of Bitcoin in the year 2008, the world has moved to the new era of technology where the control from the centralized server can be distributed over the whole network consisting of multiple peers and having the updated system knowledge. Blockchain is a highly reliable distributed record-keeping technology as it keeps track of the transaction made over the system which is digitally signed and connected to through cryptographically secure hash functions like SHA256 in the form of blocks that are chained with the previous blocks (if present) or with the genesis block, the first block of a Blockchain.

In DLT, the information is stored in a peer-to-peer network where the data is added to the ledger with the validation and consensus of the peers as in the case of a public blockchain network through

PoW/PoS which is unlike that of traditional client-server architecture. With this design, the Blockchain network is resistant to any modification made to its stored data thereby making it immutable.

## 2.2 Ethereum

Ethereum is an open-source, blockchain-based platform that helps build decentralized applications (Dapps). It features scripting functionalities. These scripting features, which are called smart contracts, are logic that operates on the network of computers in the blockchain and is responsible for the interaction of Dapp with blockchain in the form of transactions. By definition, Blockchain is a public ledger, so achieving data privacy using blockchain in its current form is not possible. Data privacy, however, is crucial for any application looking to gain its user's trust, and hence the full potential of blockchain and, in extension, decentralization could not be harnessed.

In the era of Blockchain 2.0, a Turing complete, Ethereum Blockchain technology allows the development of Dapps with the help of smart contracts. It allows the application to be built and run with a Blockchain data structure without any downtime, fraud and interference from third parties. Ethereum has its wallet and cryptocurrency called ether with wei as the smallest unit. The architecture of the ethereum Blockchain is in alignment with the traditional bitcoin architecture. It provides the scripts to be installed in the network in the form of smart contracts which describes the sequence of steps or procedure to be followed. The Ethereum network works with TCP based connection through the transport layer of the system and designed to work with PoW which allows mining by solving a computationally challenging puzzle or PoS which allow mining or validation of a block on the Blockchain depending upon the initial ether count of the miner at the consensus layer.

## 2.3 Smart Contracts

A smart contract is a self-executing contract with the terms of the agreement between buyer and seller being directly written into lines of code. The code and the agreements contained therein exist across a distributed, decentralized blockchain network. The code controls the execution, and transactions are trackable and irreversible. Smart contracts permit trusted transactions and agreements to be carried out among disparate, anonymous parties without the need for a central authority, legal system, or external enforcement mechanism. While blockchain technology has come to be thought of primarily as the foundation for bitcoin, it has evolved far beyond underpinning the virtual currency.

## 2.4 Solidity

Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs which govern the behavior of accounts within the Ethereum state. Solidity is a curly-bracket language designed to target the Ethereum Virtual Machine (EVM). It is influenced by C++, Python and JavaScript. Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features. With Solidity you can create contracts for uses such as voting, crowdfunding, blind auctions, and multi-signature wallets.

## 2.5 Metamask

MetaMask is a software cryptocurrency wallet used to interact with the Ethereum blockchain. It allows users to access their Ethereum wallet through a browser extension or mobile app, which can then be used to interact with decentralized applications. MetaMask is developed by ConsenSys Software Inc., a blockchain software company focusing on Ethereum-based tools and infrastructure.

## 2.5 ABI

ABI (Application Binary Interface) in the context of computer science is an interface between two program modules, often between operating systems and user programs.

EVM (Ethereum Virtual Machine) is the core component of the Ethereum network, and smart contracts are pieces of code stored on the Ethereum blockchain which are executed on EVM. Smart contracts written in high-level languages like Solidity or Vyper need to be compiled in EVM executable bytecode; when a smart contract is deployed, this bytecode is stored on the blockchain and is associated with an address. For Ethereum and EVM, a smart contract is just this sequence of bytecodes. To access functions defined in high-level languages, users need to translate names and arguments into byte representations for byte code to work with it. To interpret the bytes sent in response, users need to convert back to the tuple of return values defined in higher-level languages. Languages that compile for the EVM maintain strict conventions about these conversions, but in order to perform them, one must know the precise names and types associated with the operations. The ABI documents these names and types precisely, easily parsable format, doing translations between human-intended method calls and smart-contract operations discoverable and reliable.

It is very similar to API (Application Program Interface), a human-readable representation of a code's interface. ABI defines the methods and structures used to interact with the binary contract, just like API does but on a lower-level. The ABI indicates the caller of the function to encode the needed information like function signatures and variable declarations in a format that the EVM can understand to call that function in bytecode; this is called ABI encoding. ABI encoding is mostly automated, taken care of by compilers like REMIX or wallets interacting with the blockchain. Contract ABI is represented in JSON format. There are clear specifications of how to encode and decode a contract ABI.

## 2.6 Remix

Remix, more commonly known as Remix IDE, is an open-source Ethereum IDE we can use to write, compile and debug Solidity code. As such, Remix can be a hugely important tool in Web3 and Dapps development.

## 2.7 jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

## 2.8 Bootstrap

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains HTML, CSS and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

## 2.9 Web3js

Web3.js is a collection of libraries which allow you to interact with a local or remote ethereum node, using a HTTP or IPC connection. The web3 JavaScript library interacts with the Ethereum blockchain. It can retrieve user accounts, send transactions, interact with smart contracts, and more.

## 2.10 Http-Server

Http-server is a simple, zero-configuration command-line static HTTP server. It is powerful enough for production usage, but it's simple and hackable enough to be used for testing, local development and learning.

# Chapter 3

# Implemented Model

## 3.1 Initialization Phase

In this phase, election authorities deploy a smart contract and host the main application. A voter can interact with the system only through this updated application as this contains the new contract address. Also, a candidate list is provided initially with the deployed contract.

For scalability, we suggest a separate set of contracts for each district, but we intend to discuss and outline our system with respect to one district, and the same process can be followed for any other. We also assume that each voter has an Ethereum account needed to interact with the system. Ethereum accounts are required only because we are basing this project on the ethereum blockchain. If any other blockchain was used, it accordingly could be integrated.

## 3.2 Verification Phase

This phase begins as soon as the initialization phase is over. In this phase, we expect eligible voters to log in using their metamask wallets. For the time being, the verification process of a voter is simply the metamask wallet address.

## 3.3 Voting Phase

This is the final phase of the system and is performed on the ethereum blockchain as a transaction. In this phase, each voter using his wallet will transact a vote for a specific candidate. The voting transaction will be successfully done only if the user is a valid voter, which means he/she has not cast any vote previously. After a successful transaction, a small amount of ether as the gas price will be deducted from the voter's wallet.

The system activity sequence diagram, which shows the interaction among phases mentioned in the previous section, can be seen in figure 1.

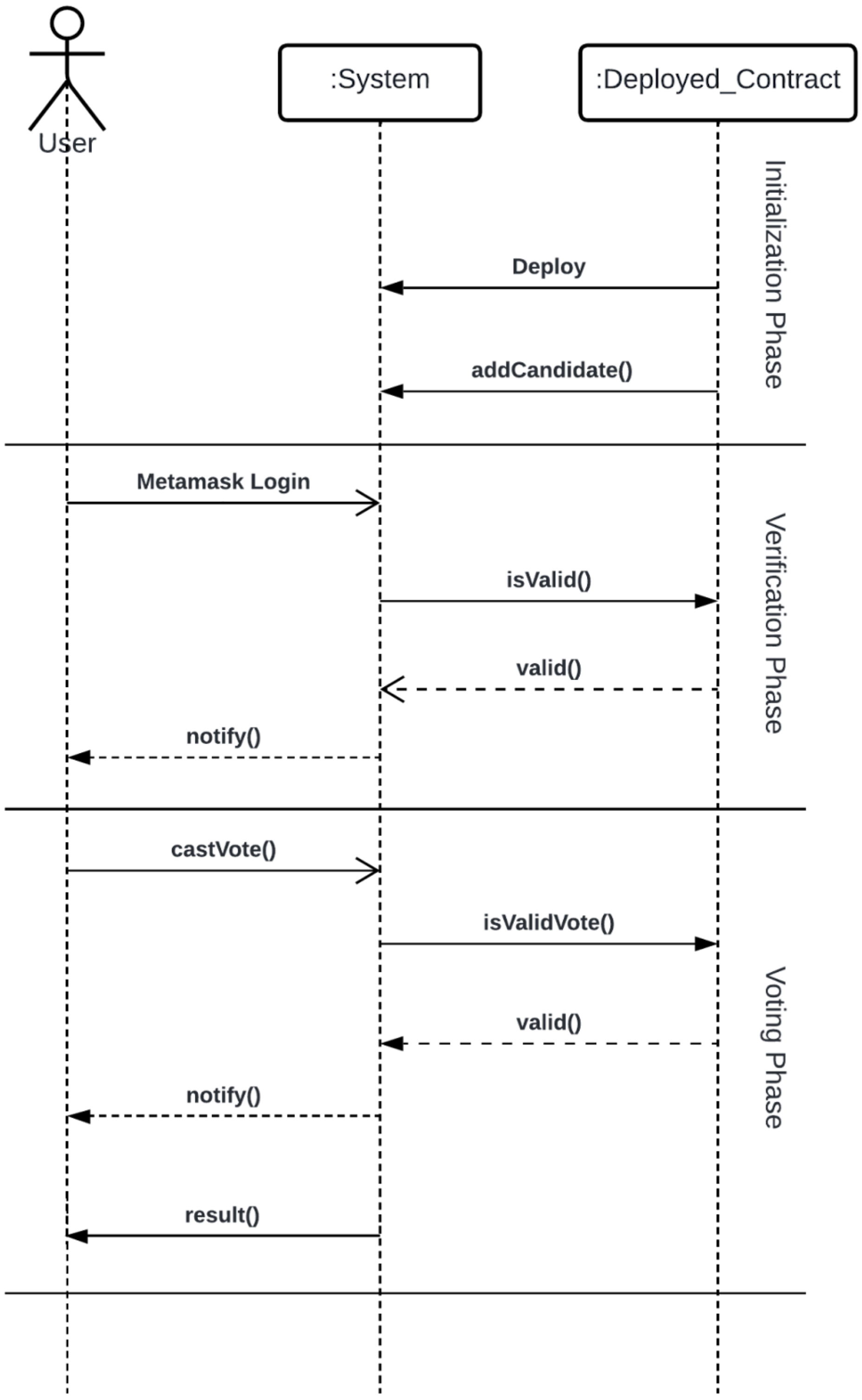


*Fig 1: Sequence diagram of the application.*

We are using Rinkeby Ethereum, a testnet version of Ethereum-based blockchain. It provides the same environment as the mainnet of blockchain; only it does not cost money. It is a public blockchain, and it uses the Proof of Work (PoW) Algorithm. Access to this testnet can be achieved through Remix online IDE, which is used to create, compile, deploy, and interact with smart contracts. It needs a wallet like metamask to support transactions with smart contracts. We have tested our smart contract using this IDE with the Rinkeby Testnet and test ether.

Below, Algorithm 1 provides pseudo-code for the Adding Candidate functionality constructed by the OOP method.

## 3.4 Algorithm 1: Add Candidate List (Through Constructor)

**Procedure:**

```
addCandidate(Candidate Name, Candidate Age, Candidate Sex)
candidate[index] = Candidate_Structure(index,  Candidate Name, Initial Votes)
Index = index + 1
```

The structure of a Candidate is consists of

- Candidate ID
- Candidate Name
- Candidate Age
- Candidate Sex
- Candidate Vote Count

Also, algorithm 2 provides the pseudo-code for the Casting Vote functionality.

## 3.5 Algorithm 2: Casting Vote

**Procedure:**

```
castVote(Candidate Index)
require(status[voter] == false)
Candidate_Structure candidate = candidate[index]
candidate.Candidate Vote Count = candidate.Candidate Vote Count + 1
status[voter] = true
```

## 3.6 Algorithm 3: Add Candidate (Through Button)

**Procedure:**

```
addCandidateButton(Candidate Name, Candidate Age, Candidate Sex)
require(message.sender==owner)
addCandidate(Name,  Age,  Sex)
```

Also, the Smart contract provides the following functionality to support different phases of the election process including above.

- Add a Candidate (The call to this can only go through while the deployment of the contract)
- Vote Casting (Only accessible by the valid voters)
- Fetch vote count of individual candidate
- Fetch status of a particular voting address
- Show results and verify Vote.
- Generating dynamic report about every aspect of the election.

# Chapter 4

# Simulation and Result

For our implementation, we decided to use Ethereum Blockchain because it is one of the widely used open source platforms for developing and deploying decentralized applications. Ethereum provides a wide range of services and solutions with its readily available development tools and smart contracts. A smart contract is a self-executing program that runs on the blockchain. In order to get compiled and deployed on the blockchain, each node on the network then executes the contract in exchange for some ether. The currency required to execute a contract is called the gas amount and varies from contract to contract.

All the nodes on the Ethereum blockchain operate in real-time. This ensures that every transaction that happens is verified by all the nodes or no nodes at all. If there is a discrepancy at a node in the network, all the surrounding nodes drop the contract and create a fork in the network. This creates a divergence in the blockchain which is discarded. Blockchain maintains a ledger that is completely online hence, it cannot be tampered with. If anyone were to change it, the hoax node would get rejected.

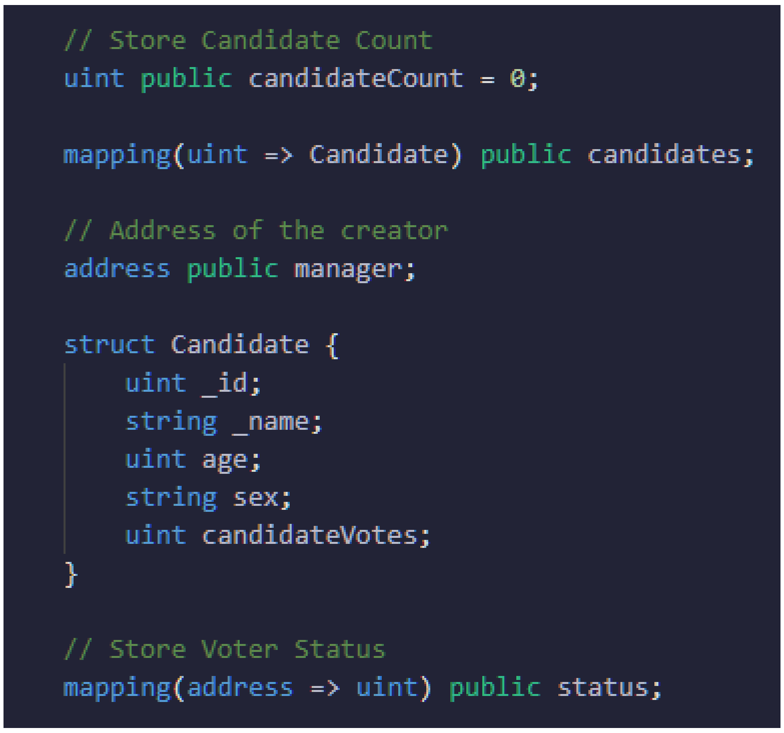
The structure that we will be using here is similar to an object-oriented approach. For us to make it an easy process, we need to make sure of a few things. First, we need transparency. Thanks to blockchain, all the records on the network are available to everyone. Second, we need to eliminate the problem of fake votes. For this, we ensure that every vote is real and is done by a real person on the blockchain. Also, a person should be able to vote only once. All of these issues can be resolved if we use blockchain in the backend as an underlying layer. Due to the immutable nature of the blockchain, if something is initialized, it can never be altered.

The language of choice will be solidity which is a programming language that runs directly on the blockchain. The smart contracts are written in solidity as well. Contracts are executed by all the nodes and the updated information is shared amongst other nodes after a regular interval. These contracts should be validated by at least 2 nodes to become activated. Even though the Ethereum blockchain is free to use, it costs ether for nodes to execute a smart contract. This cost is referred to as gas. In Ethereum, gas varies based on the smart contract and its functions. Smart contracts also don't require proof of work as each node will perform transactions with the contract. Since Ethereum's main network deals with real Ethereum which is costly, we decided to use the Rinkeby Test Network. Rinkeby test network is widely used to test solidity smart contracts using self-provide testing ether, which can be collected through the available public faucets.
We have mainly collected ethers from Rinkeby Faucet[16] and Chainlink Faucets[17]. Both of the sources provide free test ethers to a specific metamask wallet address for an interval of 12-24 hours per request.

In our code, we have a contract 'DecentralizedElectionSystem' which is similar to a class in C++ or Java. It has all the components required in our smart contract. To define each candidate standing in the election, we are using a complex data type struct that has the candidate's ID, his Name and his Vote count. Because of the nature of solidity, all these values are set to null or 0 by default.

Once we have declared the candidate, we need to map candidates to voters so that one voter can only vote for a single candidate. Then, we need to store all the candidates that are in the election. Finally, we need to check and give permission to vote for only those people who still have to vote.

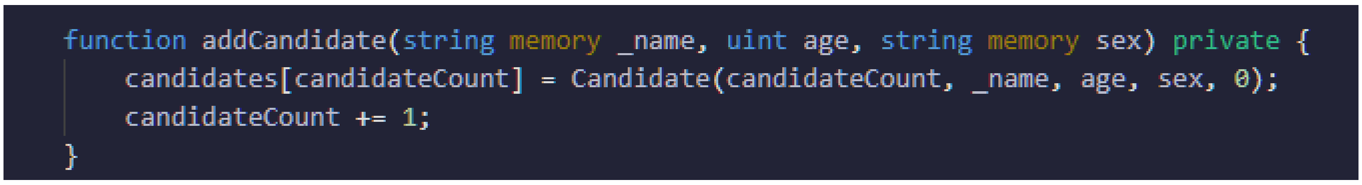
To accomplish this, we will use a require() function that will only allow a voter to vote once.



```
// Store Candidate Count
uint public candidateCount = 0;

mapping(uint => Candidate) public candidates;

// Address of the creator
address public manager;

struct Candidate {
    uint _id;
    string _name;
    uint age;
    string sex;
    uint candidateVotes;
}

// Store Voter Status
mapping(address => uint) public status;
```

*Fig 2: Code block to map candidates and voter status.*

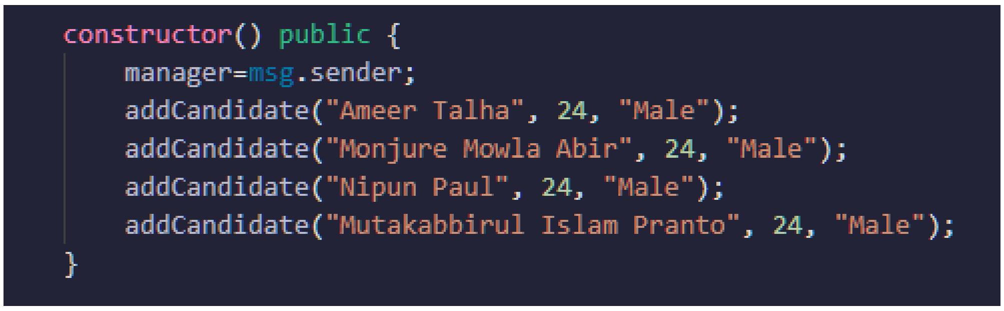
Now that all the definitions are out of the way, we will be adding functions to our contract which will help the user to vote. But before we start to vote, we must add candidates. For this, we will be using two separate functions. One is private function addCandidate(), and the later one is addCandidateButton(). The addCandidate() function will add a new candidate to the blockchain and will increment the number of candidates by one when deploying the smart contract so that it is easier for our API to list out all the candidates.



```
function addCandidate(string memory _name, uint age, string memory sex) private {
    candidates[candidateCount] = Candidate(candidateCount, _name, age, sex, 0);
    candidateCount += 1;
}
```

*Fig 3: Code block for adding a candidate to the list through constructor.*

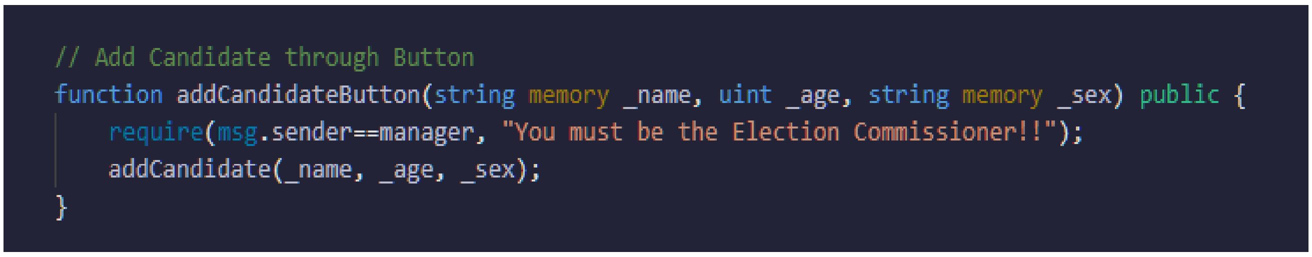
We add a constructor method to our code to make it easier to call the addCandidate() method whenever it is executed by any of the nodes.



```
constructor() public {
    manager=msg.sender;
    addCandidate("Ameer Talha", 24, "Male");
    addCandidate("Monjure Mowla Abir", 24, "Male");
    addCandidate("Nipun Paul", 24, "Male");
    addCandidate("Mutakabbirul Islam Pranto", 24, "Male");
}
```

*Fig 4: Code block for defining a constructor.*

The second method, addCandidateButton(), can be called dynamically only by the owner of the smart contract, that is, the election commissioner. On the aspect of others, the transaction will be reverted because of the require() function.



```
// Add Candidate through Button
function addCandidateButton(string memory _name, uint _age, string memory _sex) public {
    require(msg.sender==manager, "You must be the Election Commissioner!!");
    addCandidate(_name, _age, _sex);
}
```

*Fig 5: Code block for adding a candidate through button.*

Lastly, we add the castVote() function which will allow our voters to vote for their preferred candidate. To identify which candidate a voter is voting for, we will read the candidate's unique ID as an index which we declared in the beginning. When the user selects and votes for a candidate, he/she has used their voting privileges and cannot vote again. To make sure this happens, we check for it by using a special require() function for the user address using mapping called status[msg.sender]. This function takes in the address of the person calling our function vote and then checks with our mapping to see if the user has already voted or not. If the user has already voted, the require() will make sure that the rest of the code is not executed at all and the user isn't allowed to vote twice.

This takes away a lot of power from the user which he/she might use to cast fake votes and increase a particular candidate's count. Before voting, the user can see the total number of votes each candidate has. After voting, the window updates with the updated result.

To carry out the voting, users need to pay a small gas amount. This gas payment can be done in multiple ways. One of the ways that we will be using here is Metamask. It is an extension for Google Chrome or Mozilla Firefox. Metamask allows us to visit the distributed blockchain through our browser and run Ethereum contracts without running the full Ethereum node. This is perfect for our app here because we are only concerned with the voting app and not with the other heavy transactions on the blockchain. To vote using metamask, we must create an account and connect to the test network using our given address.

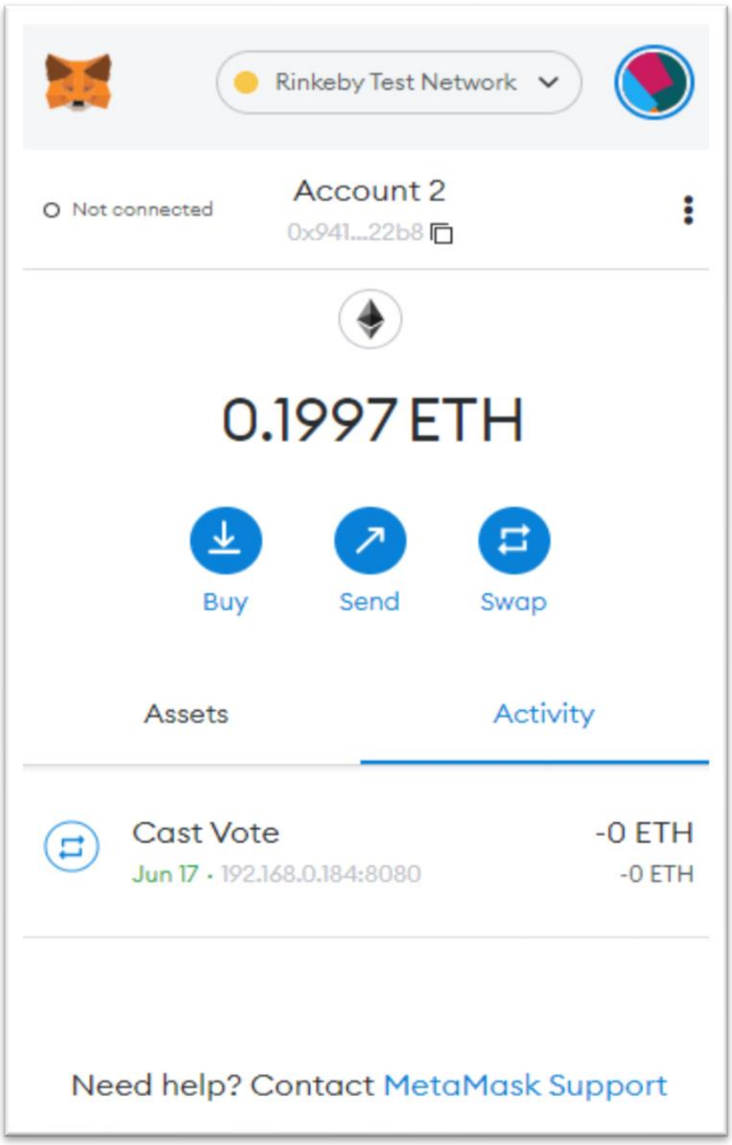
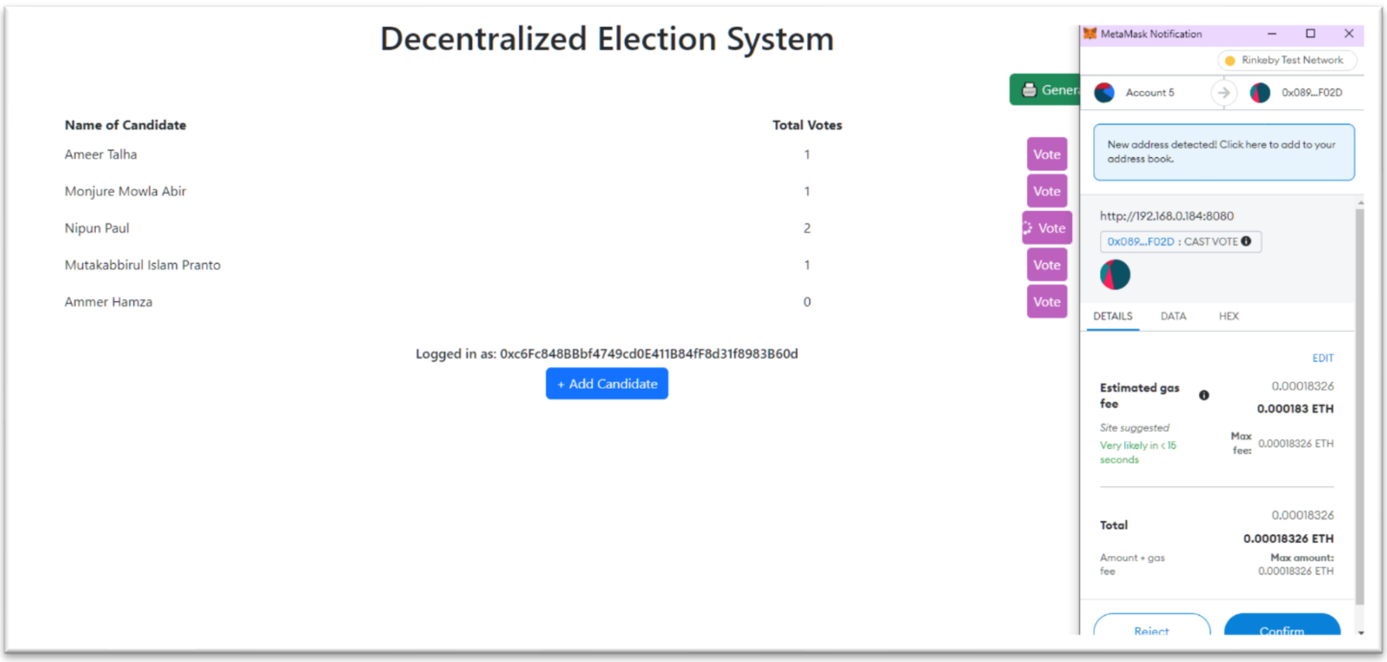


*Fig 6: Metamask.*

To connect to our application first, we have to install the http-server package which is available on node.js. After installing, we've to go to the directory where the actual frontend file resides and run http-server. The actual file, index.html, consists of Html, Bootstrap, jQuery and Javascript, where jQuery builds the bridge between the deployed contract and our frontend application through Web3.js.

Once the page loads, we log in to our metamask account and connect to the test network we are running on.

Voting on the blockchain uses a fraction of our ether as gas amount. Once the gas is paid, the transaction occurs and all the other nodes get to know about this transaction as well so the voting occurs in real-time and becomes difficult to tamper with.


Decentralized Election System

| Name of Candidate | Total Votes |
|---|---|
| Ameer Talha | 1 |
| Monjure Mowla Abir | 1 |
| Nipun Paul | 2 |
| Mutakabbirul Islam Pranto | 1 |
| Ammer Hamza | 0 |

Logged in as: 0xc6Fc848BBbf4749cd0E411B84fF8d31f8983B60d

MetaMask Notification

Rinkeby Test Network

Account 5 → 0x089...F02D

New address detected! Click here to add to your address book.

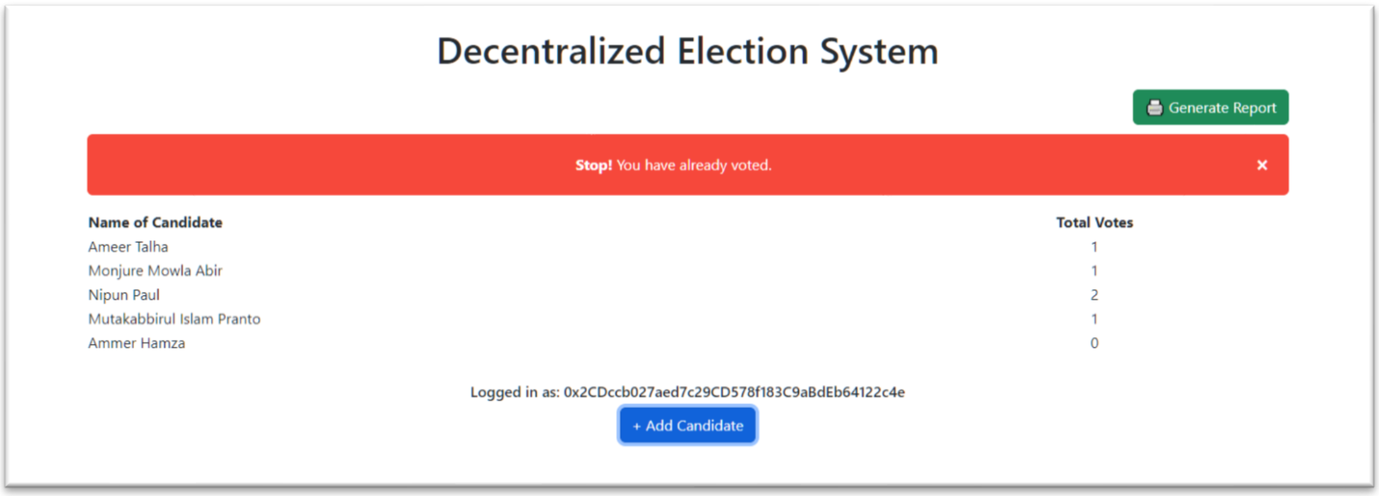
http://192.168.0.184:8080

0x089...F02D : CAST VOTE

| | |
|---|---|
| Estimated gas fee | 0.00018326<br>0.000183 ETH |
| Site suggested<br>Very likely in < 15 seconds | Max fee: 0.00018326 ETH |
| Total | 0.00018326<br>0.00018326 ETH |
| Amount + gas fee | Max amount: 0.00018326 ETH |

*Fig 7: Transaction request from metamask while casting a vote.*

Since a user can vote only once, after voting, he/she doesn't get the interface to vote again instead, they get to see the results of the election.


Decentralized Election System

**Stop!** You have already voted.

| Name of Candidate | Total Votes |
|---|---|
| Ameer Talha | 1 |
| Monjure Mowla Abir | 1 |
| Nipun Paul | 2 |
| Mutakabbirul Islam Pranto | 1 |
| Ammer Hamza | 0 |

Logged in as: 0x2CDccb027aed7c29CD578f183C9aBdEb64122c4e

*Fig 8: While casting a second vote from the same address.*

These are the functionalities of our project that we have implemented by using blockchain.

As we have implemented our Decentralized Election System application, we have faced some challenges during our working process. These challenges hampered our workflow and time but for these, but we learned and improved many things by tackling them.

Firstly, our application is based on Blockchain technology which is a new platform that is rapidly gaining momentum in this era. As it is a new technology the lack of resources for developing an app using it is very challenging for us. In our project, we have implemented Bootstrap, HTML, CSS and jQuery for the front-end part. But it was so difficult to manage resources for that particular frontend framework. This was a huge challenge for us to proceed through our workflow of the development of this app. As the resources were very minimal, we had to figure out and do it ourselves to solve the problem. For this, we took a lot of time developing and debugging it which we didn't expect.

Secondly, we have chosen Bootstrap and jQuery to build our front-end part of the application which is fair if we want to keep a less complex website for our project. But if we wanted to change it and make it a complex, multi-page website then we have to repeat those front-end parts again and again which is a problem for our project. However, the code will be huge as we are working on just a simple website. For this, error checking, handling, or debugging will be very hard.

Finally, our main goal of decentralizing the transactions happening with the voting system has been accomplished with the application created and we have been successful in securing the privacy as well as keeping it transparent to safeguard the decision of the users.

The entire counting process of the votes is open to all to be monitored thus, reducing any chances of manipulations of the votes and the results are seen in real-time.

We have implemented some features that are suggested by some previous works and also implemented some new features.

In our application, voters can cast votes successfully and everyone can see which candidate got how many votes. Voters have to pay some gas fee to complete the transaction for the successful casting of votes.

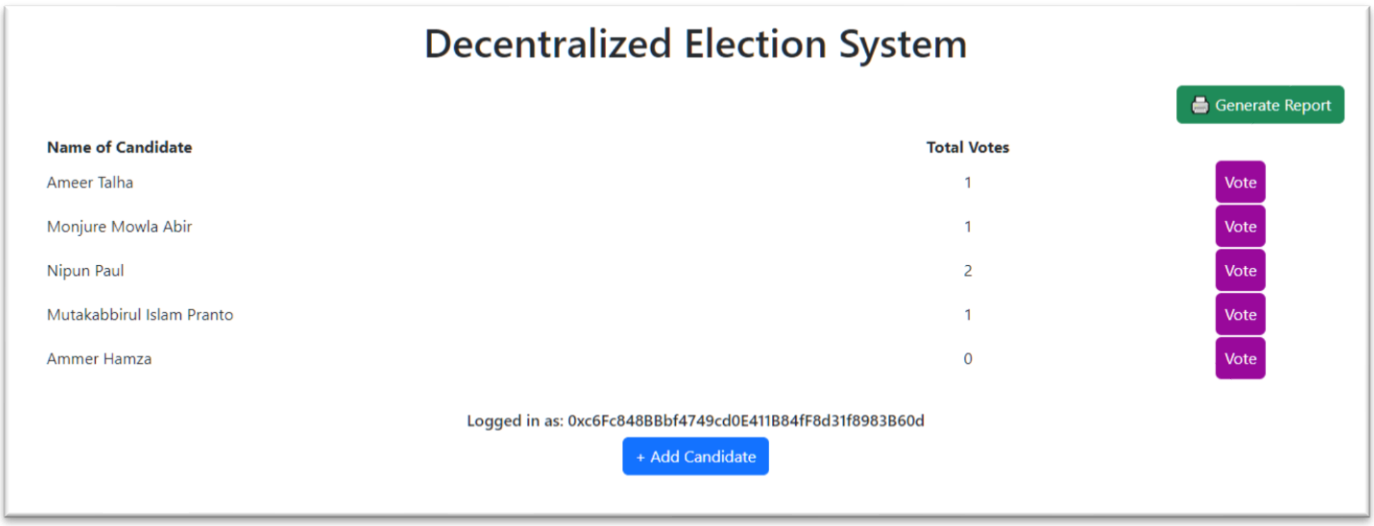
However, one voter can vote only once. So if anyone wants to vote a second time using the same address, the vote button will disappear, so one cannot cast multiple votes on one candidate.

To confirm a valid transaction, it costs some gas as well as a specific amount of time (around 14-15 minutes) to deploy the new block on the blockchain. So, we need to show the voter a loading component as well as disable casting votes on that specific time limit, which has been done.

On the other hand, we had to manually deploy our smart contract and copy the ABI to implement the frontend application, which was a little hassle.

Also, there are some minor bugs, which we were not able to solve. Most of them are related to the front-end part of our application.

So, this is the front-end application that the voters will see when they login with their metamask account and they can cast vote on whoever they want only once. After voting has been done the results will be displayed. This is how our application works.



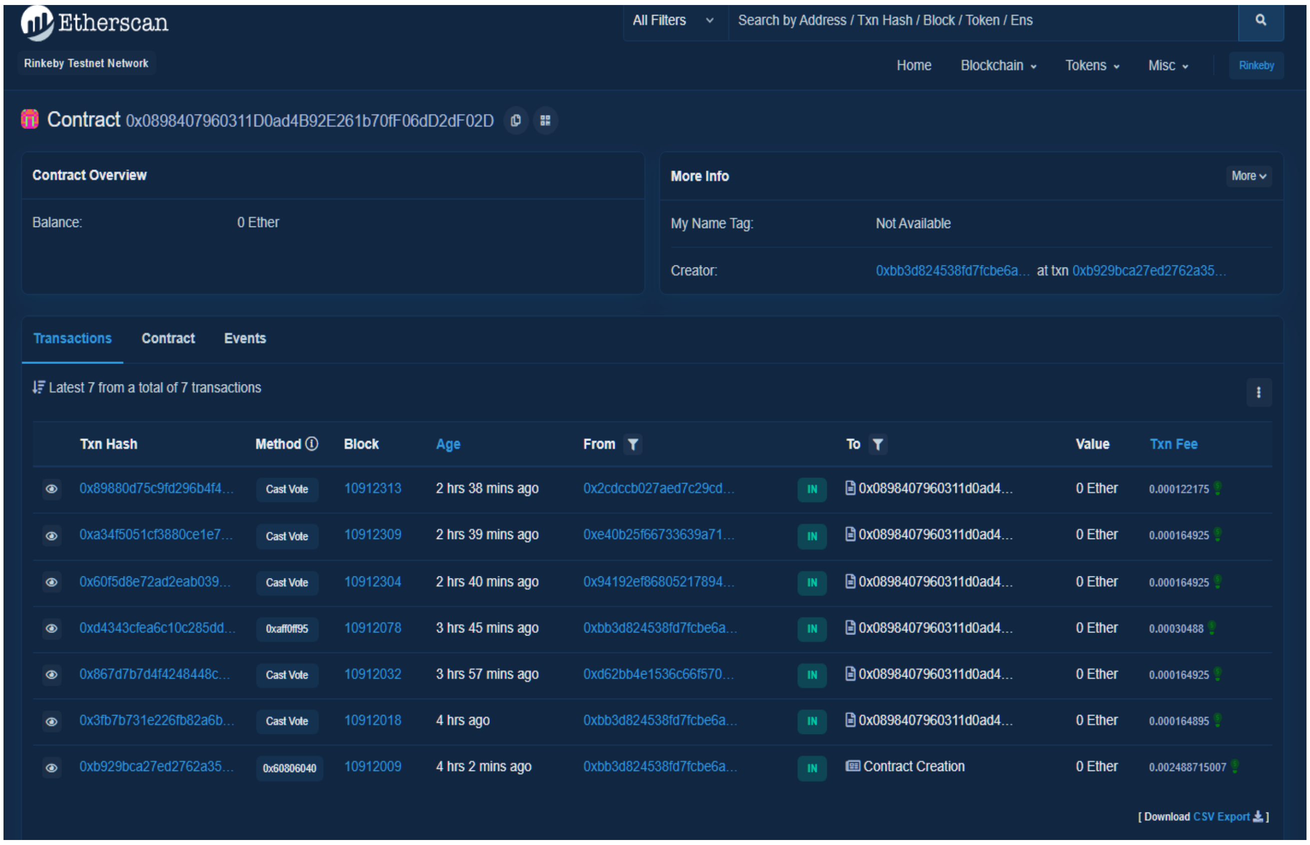
Decentralized Election System

| Name of Candidate | Total Votes | |
|---|---|---|
| Ameer Talha | 1 | Vote |
| Monjure Mowla Abir | 1 | Vote |
| Nipun Paul | 2 | Vote |
| Mutakabbirul Islam Pranto | 1 | Vote |
| Ammer Hamza | 0 | Vote |

Logged in as: 0xc6Fc848BBbf4749cd0E411B84fF8d31f8983B60d

*Fig 9: Initial view of the application.*

We can get the details of the transactions and the created blocks through Etherscan[18].



Contract 0x0898407960311D0ad4B92E261b70fF06dD2dF02D

Contract Overview

Balance: 0 Ether

More Info

My Name Tag: Not Available

Creator: 0xbb3d824538fd7fcbe6a... at txn 0xb929bca27ed2762a35...

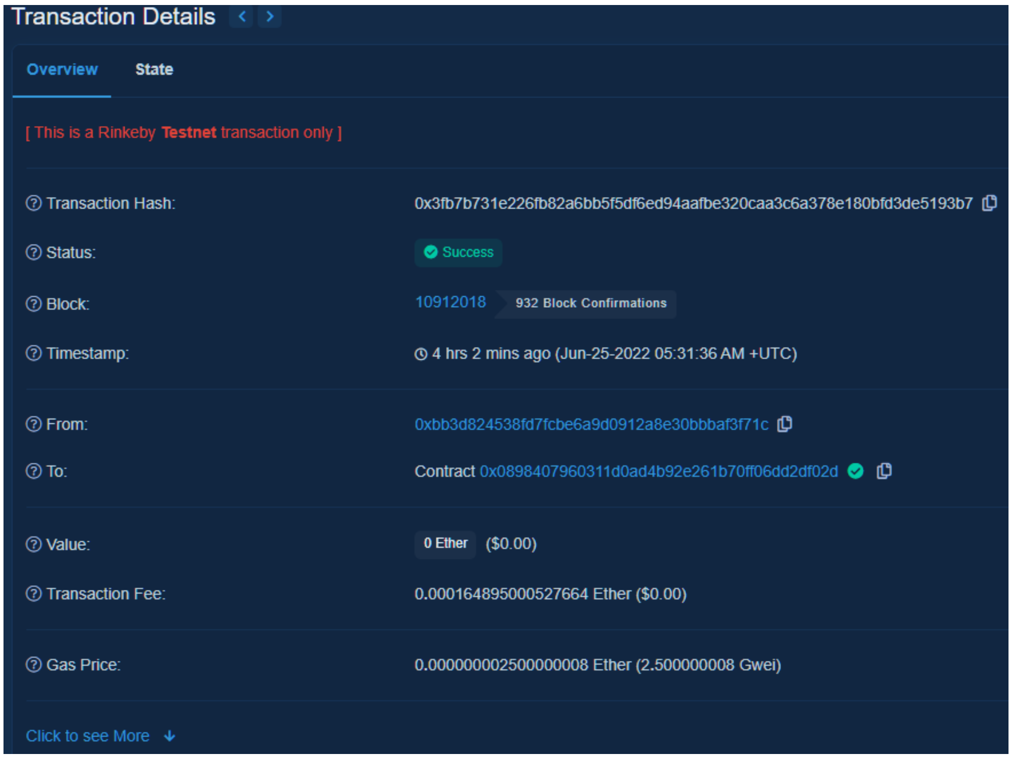
Transactions | Contract | Events

Latest 7 from a total of 7 transactions

| Txn Hash | Method | Block | Age | From | | To | Value | Txn Fee |
|---|---|---|---|---|---|---|---|---|
| 0x89880d75c9fd296b4f4... | Cast Vote | 10912313 | 2 hrs 38 mins ago | 0x2cdccb027aed7c29cd... | IN | 0x0898407960311d0ad4... | 0 Ether | 0.000122175 |
| 0xa34f5051cf3880ce1e7... | Cast Vote | 10912309 | 2 hrs 39 mins ago | 0xe40b25f66733639a71... | IN | 0x0898407960311d0ad4... | 0 Ether | 0.000164925 |
| 0x60f5d8e72ad2eab039... | Cast Vote | 10912304 | 2 hrs 40 mins ago | 0x94192ef86805217894... | IN | 0x0898407960311d0ad4... | 0 Ether | 0.000164925 |
| 0xd4343cfea6c10c285dd... | 0xaff0ff95 | 10912078 | 3 hrs 45 mins ago | 0xbb3d824538fd7fcbe6a... | IN | 0x0898407960311d0ad4... | 0 Ether | 0.00030488 |
| 0x867d7b7d4f4248448c... | Cast Vote | 10912032 | 3 hrs 57 mins ago | 0xd62bb4e1536c66f570... | IN | 0x0898407960311d0ad4... | 0 Ether | 0.000164925 |
| 0x3fb7b731e226fb82a6b... | Cast Vote | 10912018 | 4 hrs ago | 0xbb3d824538fd7fcbe6a... | IN | 0x0898407960311d0ad4... | 0 Ether | 0.000164895 |
| 0xb929bca27ed2762a35... | 0x60806040 | 10912009 | 4 hrs 2 mins ago | 0xbb3d824538fd7fcbe6a... | IN | Contract Creation | 0 Ether | 0.002488715007 |

[ Download CSV Export ]

*Fig 10: All transactions recorded for the deployed contract address.*



Transaction Details

Overview | State

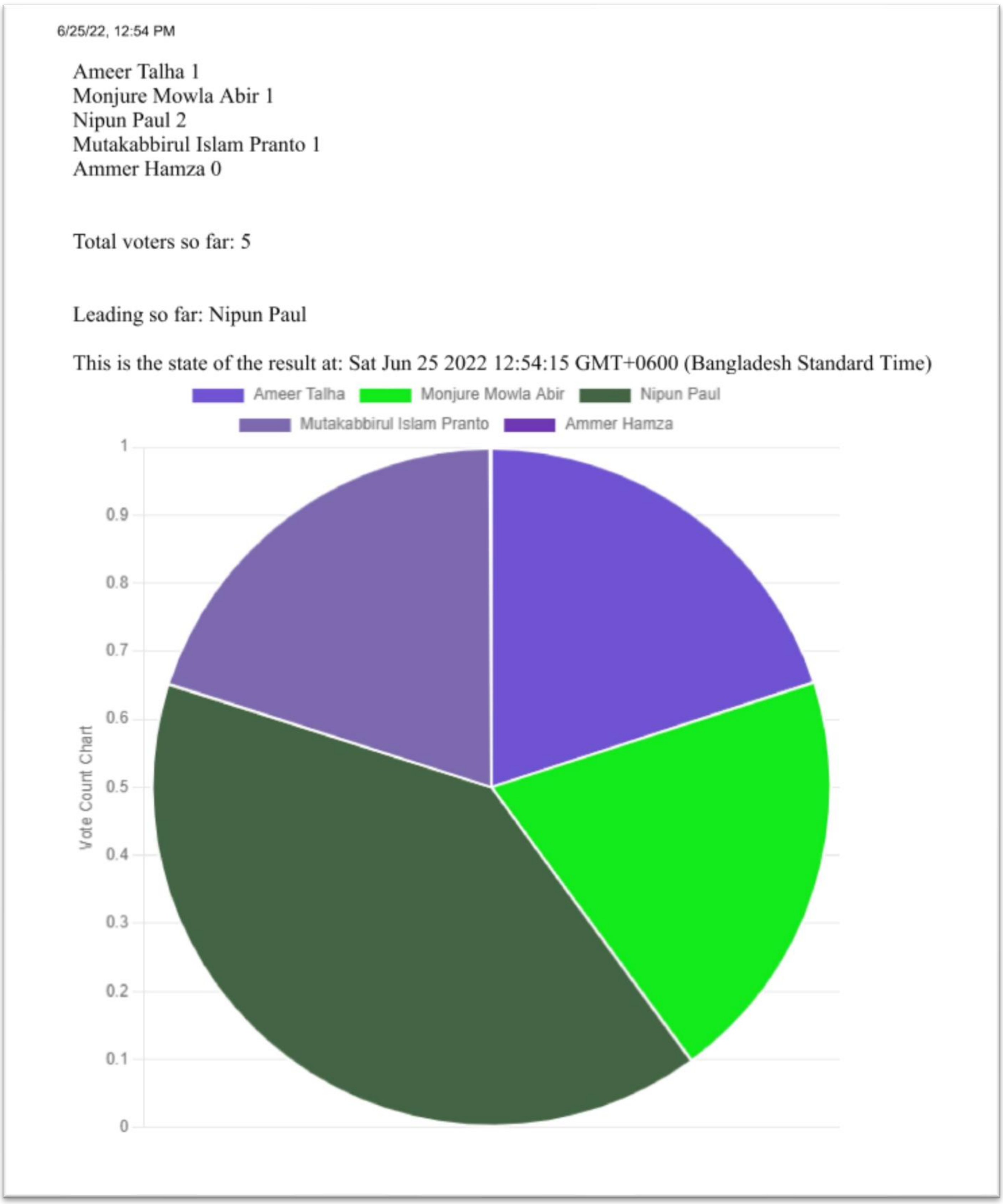
[ This is a Rinkeby **Testnet** transaction only ]

| | |
|---|---|
| Transaction Hash: | 0x3fb7b731e226fb82a6bb5f5df6ed94aafbe320caa3c6a378e180bfd3de5193b7 |
| Status: | Success |
| Block: | 10912018 932 Block Confirmations |
| Timestamp: | 4 hrs 2 mins ago (Jun-25-2022 05:31:36 AM +UTC) |
| From: | 0xbb3d824538fd7fcbe6a9d0912a8e30bbbaf3f71c |
| To: | Contract 0x0898407960311d0ad4b92e261b70ff06dd2df02d |
| Value: | 0 Ether ($0.00) |
| Transaction Fee: | 0.000164895000527664 Ether ($0.00) |
| Gas Price: | 0.000000002500000008 Ether (2.500000008 Gwei) |

Click to see More

*Fig 11: Block details of a vote casting transaction.*

The report generated from our dapp is provided below:



6/25/22, 12:54 PM

Ameer Talha 1
Monjure Mowla Abir 1
Nipun Paul 2
Mutakabbirul Islam Pranto 1
Ammer Hamza 0

Total voters so far: 5

Leading so far: Nipun Paul

This is the state of the result at: Sat Jun 25 2022 12:54:15 GMT+0600 (Bangladesh Standard Time)

*Fig 12: Candidate vote count pie chart.*

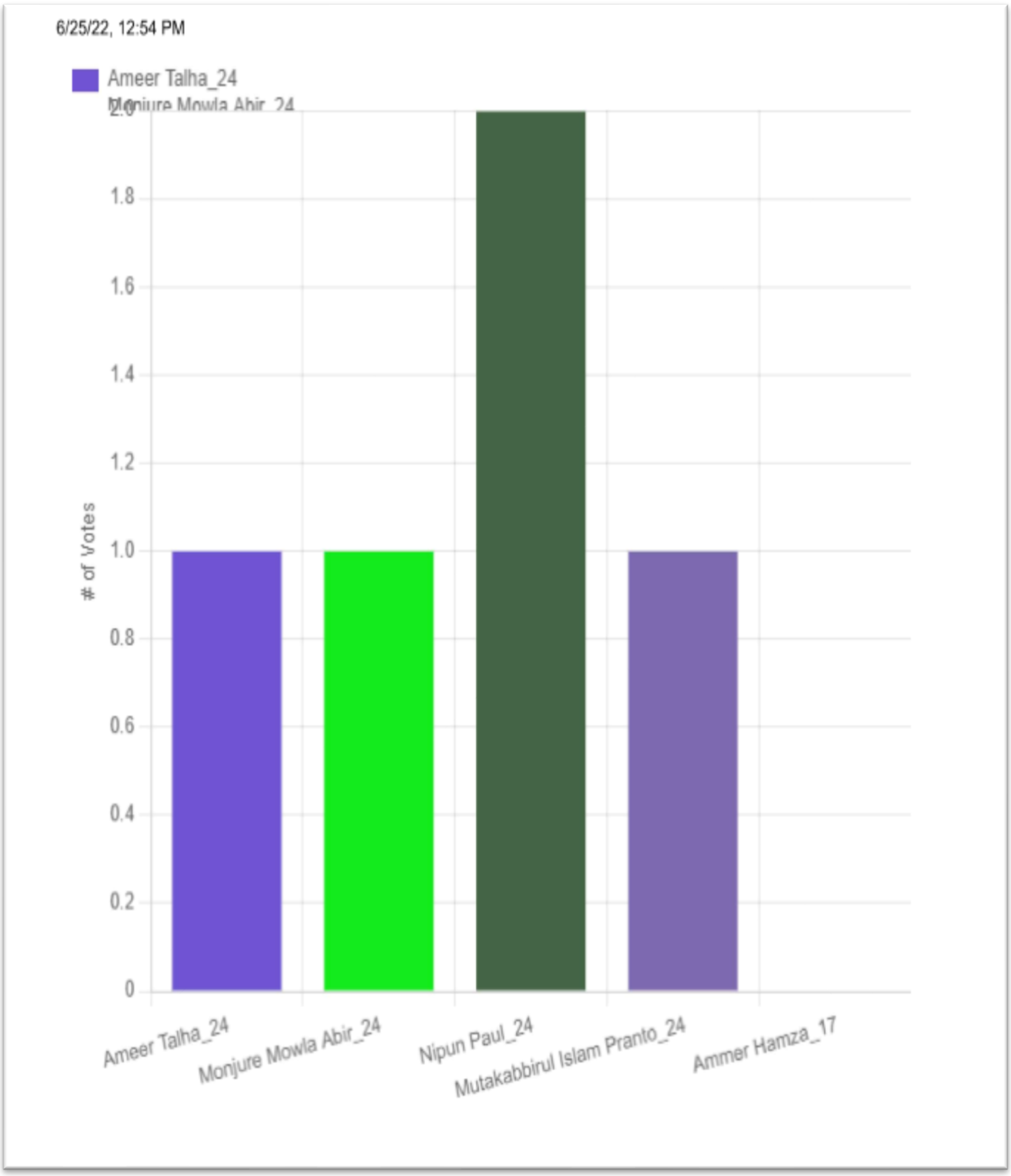


*Fig 13: Age vs vote count bar chart.*

# Chapter 5

# Conclusion

Using the smart contracts on the Ethereum Blockchain, we have created a decentralized application called the ‘Decentralized Election System’. In this project, we tried to provide a secure and comfortable experience for the voters to vote in an election. Because this project is made on the blockchain network, nobody can change any data in the whole process. Also, the voters can rest assured that their votes have been considered which is not the case for traditional voting. Therefore, we have used the properties of the blockchain technology to resolve the issues with the traditional voting system.

## Study Materials

[1] R. A. Canessane, N. Srinivasan, A. Beuria, A. Singh and B. M. Kumar, "Decentralized Applications Using Ethereum Blockchain," 2019 Fifth International Conference on Science Technology Engineering and Mathematics (ICONSTEM), 2019, pp. 75-79, doi: 10.1109/ICONSTEM.2019.8918887.

[2] A. Fernandes, K. Garg, A. Agrawal and A. Bhatia, "Decentralized Online Voting using Blockchain and Secret Contracts," 2021 International Conference on Information Networking (ICOIN), 2021, pp. 582-587, doi: 10.1109/ICOIN50884.2021.9333966.

[3] R. Taş and Ö. Ö. Tanrıöver, "Building A Decentralized Application on the Ethereum Blockchain," 2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), 2019, pp. 1-4, doi: 10.1109/ISMSIT.2019.8932806.

[4] Andreas Bogner, Mathieu Chanson, Arne Meeuw, "A Decentralised Sharing App running a Smart Contract on the Ethereum Blockchain“, 6th International Conference on the Internet of Things (IoT‘16), doi/10.1145/2991561.2998465.

[5] Divya Rathore, Virender Ranga, “Secure Remote E-Voting using Blockchain”, 5th International Conference on Intelligent Computing and Control Systems (ICICCS 2021), 10.1109/ICICCS51141.2021.9432249

[6] Eleanna Kafeza, Syed Juned Ali, Irene Kafeza, and Haseena AlKatheeri, "Legal smart contracts in Ethereum Block chain: Linking the dots", in 2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW), 10.1109/ICDEW49219.2020.00-12

[7] https://docs.soliditylang.org/en/v0.8.14/

[8] https://www.investopedia.com/terms/s/smart-contracts.asp#:~:text=A%20smart%20contract%20is%20a,a%20distributed%2C%20decentralized%20blockchain%20network

[9] https://en.wikipedia.org/wiki/MetaMask

[10] https://www.quicknode.com/guides/solidity/what-is-an-abi

[11] https://moralis.io/remix-explained-what-is-remix/

[12] https://jquery.com/

[13] https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)

[14] https://medium.com/coinmonks/web3-js-ethereum-javascript-api-72f7b22e2f0a

[15] https://www.npmjs.com/package/http-server

[16] https://rinkebyfaucet.com/

[17] https://faucets.chain.link/rinkeby

[18] https://etherscan.io/