In-house Training Report

on

**DECENTRALIZED VOTING SYSTEM USING ETHEREUM BLOCKCHAIN**

Submitted in the partial fulfillment of the requirement for the award of the degree of

**Bachelor of Technology**

**(Computer Science and Engineering)**

[2016-2020]



Under the Supervision of : Dr. Deepak Sinwar

by

1. Rahul Joshi [02610402716]

2. Aditya Sahu [00210402716]

3. Chirag Arora [41210402716]

*Department of Computer Science and Engineering*

*AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY*

(Affiliated to Guru Gobind Singh Indraprastha University, New Delhi)

[June-July, 2018]

**CERTIFICATE**

It is hereby certified that the project entitled **“Decentralized voting system using Ethereum Blockchain”** has been submitted by

1. Rahul Joshi [02610402716]

2. Aditya Sahu [00210402716]

3. Chirag Arora [41210402716],

of CSE fifth Semester, under the guidance of in-house coordinators as a part of B.Tech. in-house training (ETCS-359).

This work put in by them is an outcome of their own hard work and effort and the matter embodied in the report has not been submitted for the award of any other degree.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. Pinki Nayak

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Mr. Ram Paul

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. Deepak Sinwar

Date:

Place: Amity School of Engineering and Technology

**ACKNOWLEDGEMENT**

We take this opportunity to thank **Prof. Dr. Rekha Agarwal**, Director, who gave us the opportunity to work with the college and providing us the necessary resources.

We would also like to thank our In-house coordinators; Dr. Pinki Nayak, Dr. Deepak Sinwar and Mr. Ram Paul for their constant support during the development of the project. Their dedication and keen interest above all their overwhelming attitude to help their students had been solely and mainly responsible for completing our work. Their timely advice, meticulous scrutiny, scholarly advice and scientific approach have helped us to a very great extent to accomplish this task.

We would also like to acknowledge with much appreciation the crucial role of the staff of computer labs, Amit, Nitesh and Abhishek who gave the permission to use all required equipment and the necessary help to complete the project.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Rahul Joshi Aditya Sahu Chirag Arora

[02610402716] [00210402716] [41210402716]

**ABSTRACT**

# *There is no doubt that the revolutionary concept of the blockchain, which is the underlying technology behind the famous cryptocurrency Bitcoin and its successors, is triggering the start of a new era in the Internet and the online services. While most people focus only at cryptocurrencies; in fact, many administrative operations, and everyday services that can only be done offline and/or in person, can now safely be moved to the Internet as online services. What makes it a powerful tool for digitalizing everyday services is the introduction of smart contracts, as in the Ethereum platform. Smart contracts are meaningful pieces of codes, to be integrated in the blockchain and executed as scheduled in every step of blockchain updates. E-voting on the other hand, is another trending, yet critical, topic related to the online services. The blockchain with the smart contracts emerges as a good candidate to use in developments of safer, cheaper, more secure, more transparent, and easier-to-use e-voting systems. Ethereum and its network is one of the most suitable ones, due to its consistency, widespread use, and provision of smart contracts logic. An e-voting system must be secure, as it should not allow duplicated votes and be fully transparent, while protecting the privacy of the attendees. In this work, we have implemented and tested a sample e-voting application as a smart contract for the Ethereum network using the Ethereum wallets and the Solidity language. After an election is held, eventually, the Ethereum blockchain will hold the records of ballots and votes. Users can submit their votes directly from their Ethereum wallets, and these transaction requests are handled with the consensus of every single Ethereum node. This consensus creates a transparent environment for e-voting.*

# LIST OF FIGURES

Figure 1. Blockchain Structure 01

Figure 2. Difference between working of normal contract and smart contract. 03

Figure 3. Merkle Tree Structure. 06

Figure 4. GANTT chart 10

Figure 5. PERT chart 11

Figure 6. E-R diagram 12

**SAMPLE OUTPUTS**

Figure 7. Main page Appendix 2

Figure 8. Voting a Political Party Appendix 2

Figure 9. After transaction is successful Appendix 2

Figure 10. Final result after all the votes are voted. Appendix 2

**TABLE OF CONTENTS**

CERTIFICATE I

ACKNOWLEDGEMENT II

ABSTRACT III

LIST OF FIGURES IV

1. Introduction to the project
   1. Blockchain and Ethereum 1
   2. Smart Contracts 3
   3. Truffle 3
   4. Connecting to Ethereum Client 4
   5. web3.js 4
2. Feasibility Study
   1. Security and Reliability 5
   2. Financial Aspects 7
3. Software Requirement 8
4. GANTT Chart and PERT Chart 10
5. E-R diagram 12
6. Testing Techniques

A. Shift Left Testing 13

B. API testing 13

C. Functional Testing 13

D. Performance Testing 14

1. Conclusion 16
2. Limitations 17
3. Future Scope 18
4. Bibliography 19

Appendix-I (Program Code)

Appendix-II (Sample Output/ Screenshots)

1. **INTRODUCTION**

**A. Blockchain and Ethereum**

A blockchain is a decentralized computing architecture that maintains a growing list of ordered transactions grouped into blocks that are continually reconciled to keep information up-to-date, as shown in Figure 1.

|  |
| --- |
| Image result for blockchain structure |

Fig.1: Blockchain Structure : a Continuously Growing List of Ordered and Validated Transactions

Only one block can be added to the blockchain at a time. Each block is mathematically verified (using cryptography) to ensure that it follows in sequence from the previous block. The verification process is called "mining" or Proof of Work, which allows network nodes (also called "miners") to compete to have their block be the next one added to the blockchain by solving a computationally expensive puzzle. After the nodes compute the challenge (Merkle tree + hash of the previous block) they need to and a proof (string) which when concatenated with a challenge and hashed using SHA-256 gives an output which has specific amount of leading 0s. E.g when we combine challenge and proof and hash it, the output should have for example 40 leading 0s. This is very challenging problem because SHA-256, like other hash functions, has the property that it is impossible to compute the input for a given output, and also similar input strings have completely different hashes.

The winner then announces the solution to the entire network to gain a mining reward paid via cryptocurrency. The mining process combines cryptography, game theory, and incentive engineering to ensure that the network reaches consensus regarding each block in the blockchain and that no tampering occurs in the transaction history. All transaction records are kept in the blockchain and are shared with all network nodes. This decentralized process ensures properties of transparency, incorruptibility, and robustness (since there is no single point of failure). The Ethereum blockchain is a distributed state transition system, where state consists of accounts and state transitions are direct transfers of value and information between accounts. Two types of accounts exist in Ethereum:

(1) externally owned accounts (EOAs), which are controlled via private keys and only store Ethereum’s native value-token "ether" and

(2) smart contract accounts (SCAs) that are associated with contract code and can be triggered by transactions or function calls from other contracts.

To protect the blockchain from malicious attacks and abuse (such as distributed denial of service attacks in the network or hostile infinite loops in smart contract code), Ethereum also enforces a payment protocol, whereby a fee (in terms of "gas" in the Ethereum lexicon) is charged for memory storage and each computation executed in a contract or transaction. These fees are collected by miners who verify, execute, propagate transactions, and then group them into blocks. Testnets simulate the Ethereum network and EVM. They allow developers to upload and interact with smart contracts without paying the cost of gas. Smart contracts must pay gas for their computations on the Ethereum network. However, testnets provide free or unlimited gas. That allows developers to test contracts without having to pay real money for their execution.

Testnet nodes come in two main flavors:

1. *lightweight Ethereum nodes* used for small scale local testnets.

Ex: ethereumjs-testrpc - Useful for early stage contract development. This is what you will be using most of the time. Calls to lightweight testnet nodes complete very quickly and provide good error messages.

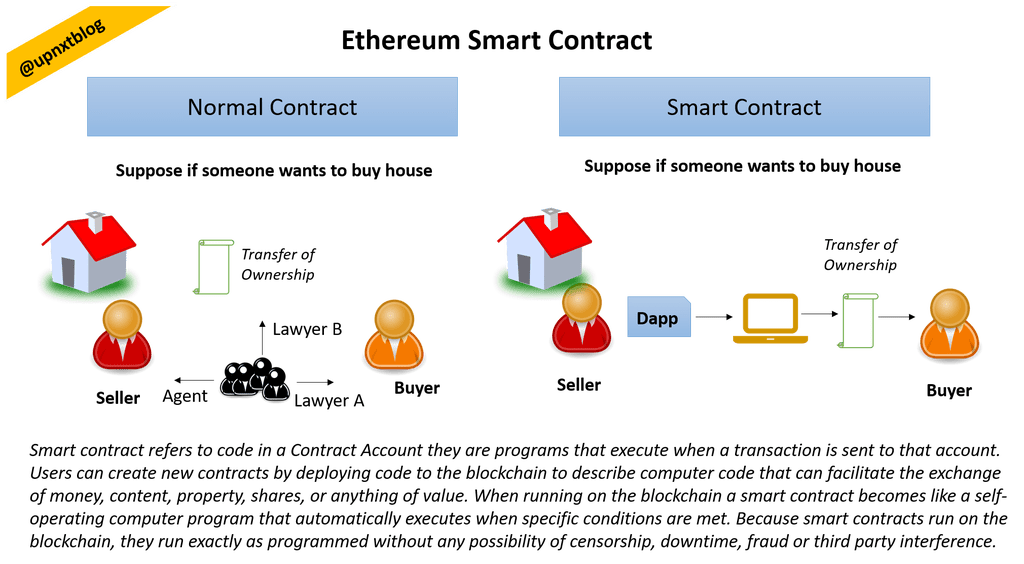
1. *heavyweight Ethereum nodes* used for large scale networked testnets.

Ex: Geth - Useful for connecting to public networked testnets. The most popular public testnet is called Ropsten which is useful during later stage contract development. connecting to Ropsten through Geth simulates the real Ethereum network. That makes it appealing for mature contracts that you want to battle test.

**B. Smart Contracts**

Ethereum smart contracts can be built in a Turing complete programming language, called Solidity. This contract language is compiled by the Ethereum Virtual Machine (EVM), which enables the Ethereum blockchain to become a platform for creating decentralized apps (DApps) that provide promising solutions to e-voting challenges.

Solidity has an object-oriented flavor and is intended primarily for writing contracts in Ethereum. A class in Solidity is realized through a "contract," which is a prototype of an object that lives on the blockchain. Contracts may contain persistent state variables that can be used as data storage and functions that interact with the states. Several smart contracts are made for the different functionalities such as returning the total votes a candidate has received and another to increment vote count for a candidate.

Fig.2: Difference between working of normal contract and smart contract.

**C. Truffle**

Truffle is a development environment, testing framework and asset pipeline for Ethereum, aiming to make life as an Ethereum developer easier. It is one of the most widely used IDEs in the Ethereum community. Developers can use it to build and deploy DApps for testing purposes with many features that make it more attractive to users with a Web 3.0 dev background. Automated contract testing with Mocha and Chai. A configurable build pipeline that supports both web apps and console apps. Generators for creating new contracts and tests (like rails generate) Instant rebuilding of assets during development (truffle watch) Console to easily work with your compiled contracts (truffle console) Script runner that lets you run JS/Coffee files with your contracts included (truffle exec) Contract compilation and deployment using the RPC client of your choice. Support for JavaScript, CoffeeScript, SASS, ES6 and JSX built-in.

**D. Connecting to Ethereum Client**

Ethereum clients expose a number of methods over JSON-RPC for interacting with them from within an application. However, interacting directly over JSON-RPC passes on a number of burdens to the application developers, such as:

1. JSON-RPC protocol implementation

2. Binary format encoding/decoding for creating and interacting with smart contracts

3. 256bit numeric types

4. Admin command support-e.g. create/manage addresses, sign transactions

**E. web3.js**

This is the Ethereum compatible JavaScript API which implements the Generic JSON RPC spec. It’s available on npm as a node module, for bower and component as an embeddable js and as a meteor.js package. web3j is a lightweight Java library for integrating with clients (nodes) on the Ethereum network. Web3js is automatically injected with the help of Mestamask or Mist.

Core features:

1. Interaction with Ethereum clients over JSON-RPC
2. Supports all JSON-RPC method types
3. Supports all Geth and Parity methods for managing accounts and signing transactions
4. Sending of client requests both asynchronously and synchronously
5. Auto-generation of Java smart contract function
6. **Feasibility Study**

This section provides results of our survey regarding the feasibility of taking advantage of blockchain technology in e-voting applications. By speaking of feasibility, we mean a cost-effective, scalable, secure, and easy-to-deploy system (or subsystem). While it is hard to determine global measures of these properties, we have considered some thresholds or factors in order to determine whether it is feasible to replace the existing (and prospective) structures with their blockchain-based counterparts, or not. Any blockchain-based solution should be (noticeably) cheaper than the traditional elections in the long term, say in a 3-years period with at least 1 elections per year. It should not be more expensive than non-blockchain solutions either. The system should support millions of people, depending on the country, business size or target group population. The security level should not be lower than non-blockchain solutions, details regarding security requirements are given later in the text.

**A. Security and Reliability**

The availability and the fault tolerance of the system is high as all the nodes keep a copy of the records and check each other to make a stable system. The blockchain provides transparency with anonymity. The privacy is not aimed but can be implemented.

Each block keeps the hash of the previous block and this eventually provides a chain of blocks that are linked to each other.

Merkle tree is used in order to keep the integrity of the records. Its structure is shown in Figure 3. Each block holds more than one transaction. Firstly, hash values of each transaction are taken and paired with the hash of the other transaction. Pairs of hashes are then combined till it results in a single root hash. Transactions can easily be verified by this structure.

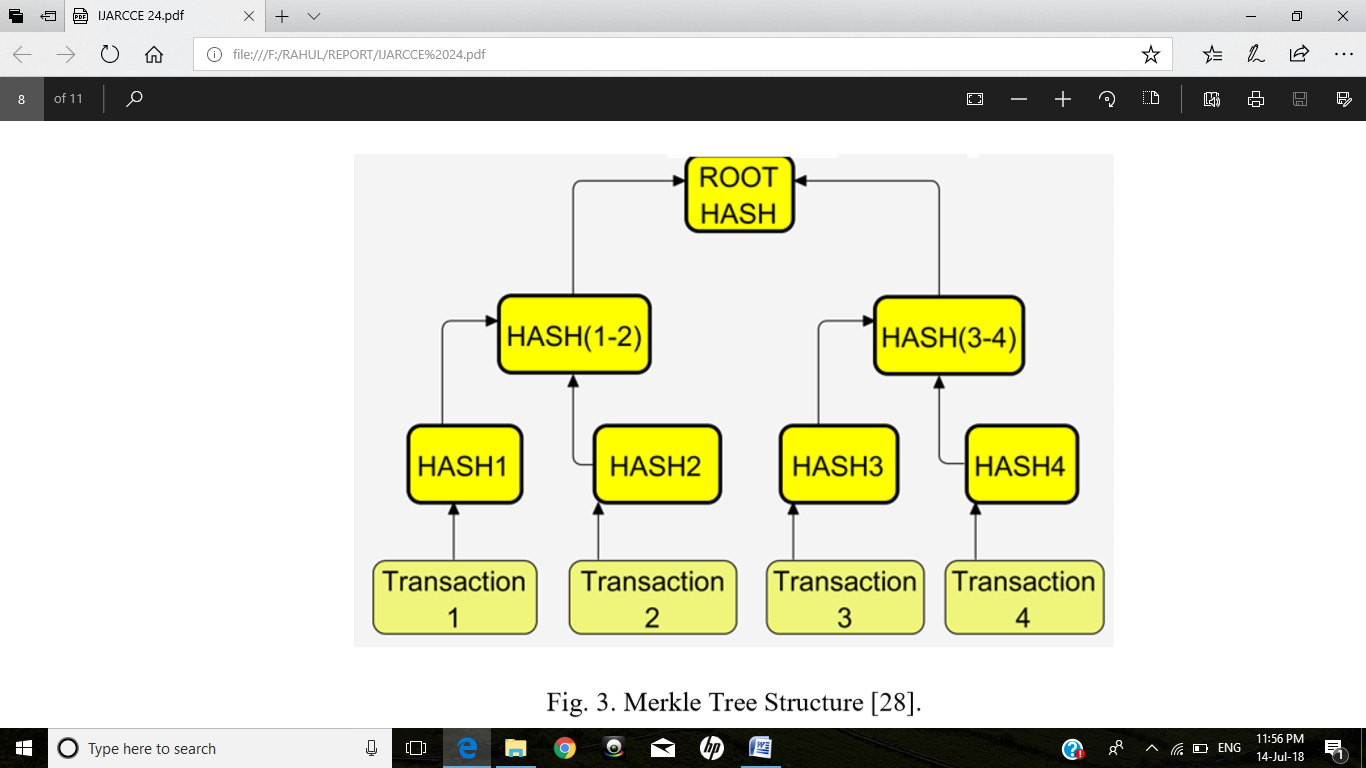


Fig. 3: Merkle Tree Structure.

Blockchain based systems are said to keep immutable records, but there is a tricky way to update the records (blocks), which is managed by the consensus protocols. Hence, the reliability of the system depends on the implemented consensus protocol. The consensus protocols (PoW, PoS, etc.) are the rules that determine which node will have the right to write to the blockchain. Bitcoin and mining-based blockchain implementations mainly use the PoW (Proof of Work) algorithm, which depends on the possessed computing power. Anyone who wants to change a block, should change that block and all of the following (next) blocks in the blockchain. For example, in a chain of 1000 blocks, if a user (or attacker) wants to change (only) the 100th block, then he/she should change all the blocks starting from 100th until 1000th. The attacker (node/computer) should get the writing turn for every block, and for PoW, this requires that it should have at least 51% of all of the computing power provided by (the sum of) all the nodes contributing to the chain. This attack is theoretically possible, especially in small networks, but as all the details of the transactions are recorded in all copies of the blockchain, it will be very easy to detect such a malicious activity; more, this negative effect can be mitigated in a short time, since it is not hard to exclude a node from the network.

The identity verification (authentication) is implemented by asymmetric cryptology in crypto coins. The public wallet address is the public key and private key is obtained from the public key. Elliptic curve cryptography (ECC) is widely used in this process. The keys are used to form Message Authentication Code (MAC) for signing the transactions. The MAC protects the integrity and authenticity of the message. These keys are also used to share the session key, which will be used to encrypt the communication for confidentiality. Additional encryption can be done for different applications. While this mechanism ensures the integrity and authentication of the user/voter accounts, authenticating the users themselves remains as an open problem.

**B. Financial Aspects**

Undoubtedly, using automated electronic systems, including web portals and mobile applications, will lower the administrative costs in the long term, despite their higher initial investment costs. A previous study showing a rough comparison regarding infrastructure and maintenance costs of traditional and electronic elections was recently made. Per to the study, the advantages of switching to an online elections system may provide savings up to several times per year. The difference becomes sharper, especially if there are 2 or more elections throughout a year. The cost from the point of view of the voters is analyzed in another work in Estonia. This study briefly suggests that the voters who live at least 30 minutes distance from their voting centers have higher (time and money) costs, and thus certainly are more likely to prefer voting online. But, elderly people’s unwillingness to use computers is neglected, as they noted.

Cost of traditional elections mainly consist of material (ballot papers, boxes etc.), personnel, and logistics costs. Yet the cost of e-voting systems includes software development, hardware infrastructure and related maintenance costs. Use of blockchain based solutions may even lower these software costs, since most blockchain bundles are open-source projects and come with customizable APIs. Besides, integrating the blockchain-based e-voting system with some sort of cryptocurrency-payment system may provide different setups and opportunities.

1. **Software Requirements**
2. visual studio code
3. It is a [source code editor](https://en.wikipedia.org/wiki/Source_code_editor) developed by [Microsoft](https://en.wikipedia.org/wiki/Microsoft) for [Windows](https://en.wikipedia.org/wiki/Windows), [Linux](https://en.wikipedia.org/wiki/Linux) and [macOS](https://en.wikipedia.org/wiki/MacOS).
4. It is based on [Electron](https://en.wikipedia.org/wiki/Electron_(software_framework)), a framework which is used to deploy [Node.js](https://en.wikipedia.org/wiki/Node.js) applications for the desktop running on the [Blink layout engine](https://en.wikipedia.org/wiki/Blink_layout_engine).
5. node.js
6. It is used for deploying smart contracts in the backend.
7. It is used for testing purpose.
8. react.js
9. It is a javascript library for building user interface.
10. web3.js
11. It is a collection of libraries which allow you to interact with a local or remote ethereum node, using a HTTP or IPC connection
12. metamask
13. It includes a secure identity vault, providing a user interface to manage your identities on different sites and sign blockchain transactions.
14. It allows to run Ethereum dApps right in your browser without running a full Ethereum node.
15. solidity
16. It is a [statically-typed](https://en.wikipedia.org/wiki/Statically_typed_programming_language) programming language designed for developing smart contracts that run on the Ethereum Virtual Machine (EVM).
17. It is used for implementing smart contractson various [blockchain](https://en.wikipedia.org/wiki/Blockchain) platforms.
18. rinkeby
19. It is a “Proof-of-Authority” network, which means that blocks are signed by well-known trusted community members. This prevents attackers from taking over the mining power on the network.
20. Rinkeby is for testing your contracts and the ethers are not real on it. It's just for testing.
21. Ganache
22. Formally known as TestRPC.
23. It is a personal blockchain for Ethereum development you can use to deploy contracts, develop your applications, and run tests.
24. Ethereum
25. It is an [open-source](https://en.wikipedia.org/wiki/Open-source), [public](https://en.wikipedia.org/wiki/Types_of_blockchain), [blockchain](https://en.wikipedia.org/wiki/Blockchain_(database))-based [distributed computing](https://en.wikipedia.org/wiki/Distributed_computing) platform and [operating system](https://en.wikipedia.org/wiki/Operating_system) featuring [smart contract](https://en.wikipedia.org/wiki/Smart_contract) (scripting) functionality
26. chrome browser
27. Google Chrome (commonly known simply as Chrome) is a [freeware](https://en.wikipedia.org/wiki/Freeware) [web browser](https://en.wikipedia.org/wiki/Web_browser) developed by [Google LLC](https://en.wikipedia.org/wiki/Google_LLC).
28. It is also the main component of [Chrome OS](https://en.wikipedia.org/wiki/Chrome_OS), where it serves as a platform for running [web apps](https://en.wikipedia.org/wiki/Web_app).
29. **GANTT CHART & PERT CHART**

* **GANTT chart**

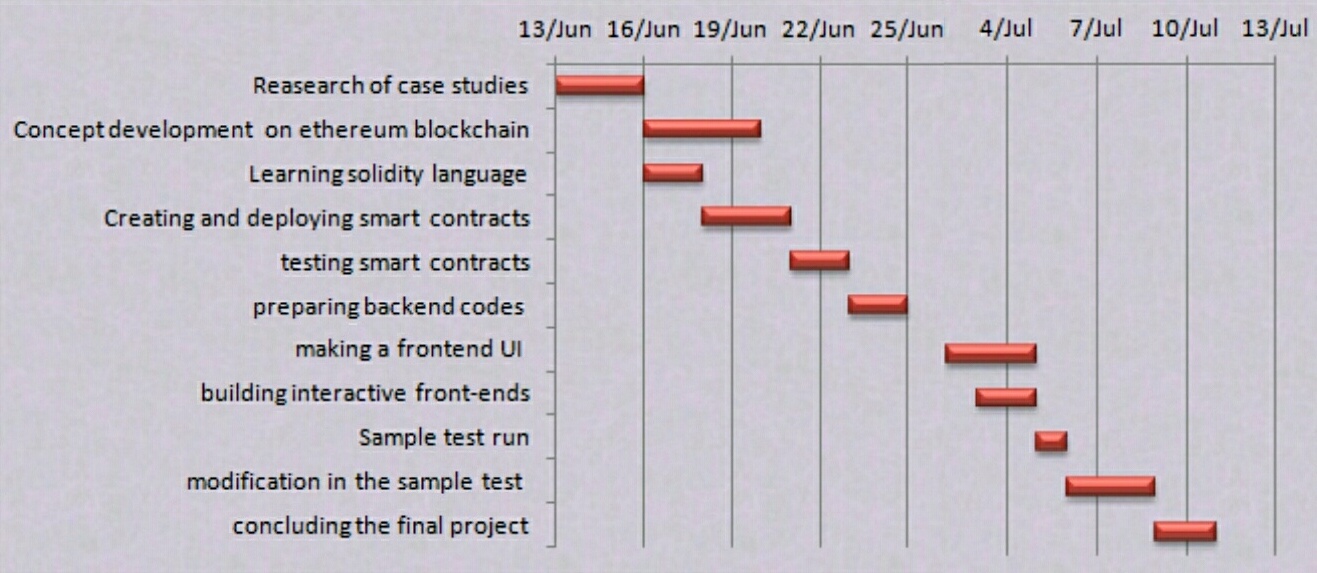
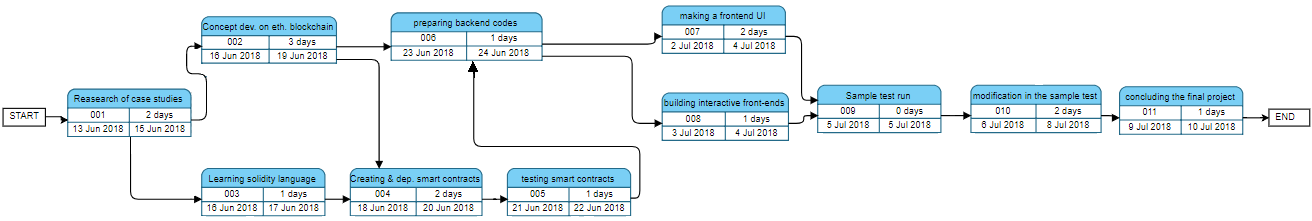
****

Fig.4: GANTT Chart for decentralized voting system

* **PERT chart**

****Fig.5: PERT chart for decentralized voting system.

1. **E-R diagram**

PARTY

1

Elects

VOTER

1

Selects

1

1

CANDIDATE

Fig.6: E-R diagram for decentralized voting system.

1. **TESTING TECHNIQUES**

To promote trust, testers need to ensure that all blockchain components work correctly and that any related applications delivered to the blockchain fabric interact in a trusted fashion. Testers should follow a best practice-based suite of testing approaches that include several paths: [Shift Left](https://www.neotys.com/insights/load-testing), [API](https://www.neotys.com/solutions/api-component-microservices), Functional, and [Performance Testing](https://www.neotys.com/insights/performance-testing).

## A. Shift Left Testing

The need for software quality in every step of the value transfer process presents a strong argument for shift left testing. More blockchain project teams are employing Agile and DevOps practices to their development and testing efforts. By shifting testing to the left, teams can perform various tests early, and reuse those tests continuously. Providing early, iterative quality feedback during development decreases the number of defects found later in the lifecycle, where the impact on the business can be severe.

## B. API Testing

Depending on the application, API testing needs to address the interaction of applications into and out of the blockchain ecosystem. Because a blockchain can emit events, external calls or events from external systems can trigger blockchain activity. Consider a transaction submitted to an API. The transaction must be validated against specific rules to generate an update order which the blockchain then distributes. The API then receives a confirmation that the blockchain has been updated. Testers must validate the interaction of applications into and out of the blockchain ecosystem at every process step to validate that API requests and responses are formatted and handled correctly.

## C. Functional Testing

As blockchain evolves into new application areas, functional testing of essential blockchain components, as well as the holistic ecosystem, is vital. Functional testing evaluates use-case scenarios and related business processes, such as the behavior of smart contracts.

Blockchain components that testers want to consider include:

1. ***Block size:*** Block size will likely vary based on the application of the blockchain. Hence, the payloads in the block will vary in size as well. For many blockchains, less than 100 bytes of each block is designated for non-payload. As of this writing, there is currently a fixed maximum limit of 1MB per block for Bitcoin. The point is, testers need to evaluate how changing block size impacts behavior. What happens when the amount of new data going into a block exceeds the maximum block size for that blockchain application. How will the block be handled? Remember that multiple transactions can be assigned to one block.
2. ***Chain size:*** The expectation of blockchain is that it provides a permanent and complete record of every value transfer transacted on the chain, so theoretically, the size of the chain is unlimited.
3. ***Data transmission:*** Due to the peer-to-peer architecture of blockchains, it’s essential to validate that the encrypted and decrypted data transmission process works flawlessly. A critical test scenario is to make sure that no data is lost.
4. ***Adding a block:*** Whenever a transaction is authenticated, testers need to validate that a new block is attached to the chain. Remember that the chain is immutable. It’s important that whenever a new block is added that it be added correctly because it can never be changed.

## D. Performance Testing

At article press time, blockchain performance is hampered by an inability to scale – as an example; blockchain is currently unable to process the volumes of transactions necessary to replace the proprietary payment processing systems of financial service organizations. Therefore, as blockchain evolves, the participant-user experience will focus as much on trust as transaction processing speed. The performance test should be applied to blockchain based on the perspective of a client app end user, responses required from smart contracts, and system interfaces.

Performance testing in [blockchain](https://www.neotys.com/resources/whitepaper/blockchain-best-practices) includes identifying performance bottlenecks, defining the metrics for tuning the system, and assessing whether the application is production ready. Testers should anticipate variances in their performance test because latency will vary by the size of the P2P network on which the blockchain exists and the volume of the transactions. Scenarios can be further influenced by data type and server locations. [Automated](https://www.neotys.com/insights/automation-testing) performance testing is key to assessing the overall scalability of the blockchain ecosystem. End-to-end scenarios need to combine all aspects of the blockchain ecosystem and should include compound testing with multiple endpoints.

Because of the shared ledger that powers the blockchain, it needs to reflect the same sequence of transactions at every node. As latencies between different consensus protocols may vary, testers need to perform peer/node testing to ensure the consistency and performance of transactions posted as new blocks. They will also need to provide that transactions post in the proper sequence to ensure that the integrity of the network and the share ledger are maintained.

1. **CONCLUSION**

By building this proposed smart contract of ours, we have succeeded in moving e-voting to the blockchain platform and we addressed some of the fundamental issues that legacy e-voting systems have, by using the power of the Ethereum network and the blockchain structure. As a result of our trials, the concept of blockchain and the security methodology which it uses, namely immutable hash chains, has become adaptable to polls and elections. This achievement may even pave the way for other blockchain applications that have impact on every aspect of human life. At this point, Ethereum and the smart contracts, which made one of the most revolutionary breakthroughs since the blockchain itself, helped to overturn the limited perception of blockchain as a cryptocurrency (coin), and turned it into a broader solution-base for many Internet-related issues of the modern world, and may enable the global use of blockchain. E-voting is still a controversial topic within both political and scientific circles. Despite the existence of a few very good examples, most of which are still in use; many more attempts were either failed to provide the security and privacy features of a traditional election or have serious usability and scalability issues. On the contrary, blockchain based e-voting solutions, including the one we have implemented using the smart contracts and the Ethereum network, address (or may address with relevant modifications) almost all of the security concerns, like privacy of voters, integrity, verification and non-repudiation of votes, and transparency of counting. Additionally, in today's connected world, with the concept of the Internet of Things (IoT), expectedly, many non-computer devices will gain access to the Internet. While we are still working on a mobile phone application as a supportive extension to our work to widen the usability; it is important to note that, apart from phones and tablets; air conditioning devices, cars, chairs, clothes, refrigerators, televisions, and many other everyday objects are/will be able to directly reach to the internet. In terms of blockchain, it won’t be difficult to build such distributed systems when there is such a large network and a reserve processing power. Moreover, if all these devices work together as a grid to shorten the validation period of transactions in a blockchain, we will be able to do most of our online transactions securely, reliably, and effectively, not only in theory but also in practice.

1. **LIMITATIONS**

1. **The process is slow**

Even ~12 seconds is unacceptable for users who are used to just clicking on stuff and having it work, and it will often be longer than that until a transaction actually gets into a block.

1. **The Blockchain architecture is expensive**

Everything you do costs a little bit of money, which means you always have to think about the cost. This will only get worse if the network gets busier.

1. **Things are to be kept public**

You can't keep any secrets or provide any privacy.

1. **The process doesn't mesh easily with the "real world"**

You need complicated mechanisms to do even the simplest actions that rely on something off-chain.

1. **Execution of complex operations**

Another pain point is the impossibility to execute complex operations on the Ethereum Virtual Machine due to some other constraints like the missing support for floating point arithmetic operations.

1. **FUTURE SCOPE**

A standardized electronic voting solution which would be widely adopted has not yet emerged, and although there are some good candidates, there are inherent security issues which make these protocols unsuitable for elections. The literature identifies a distinct gap in the domain which could be filled by a protocol, using a different technology then the previous protocols.

* 1. Blockchain offers an inherently more secure platform, and with development of the recent anonymous transaction scheme, it is finally possible to *tackle the anonymity issues* of blockchain transactions, which would open a possibility for blockchain voting.
  2. The drive behind a *cheaper, unified, electronic voting system* was the basis for the protocol, which has potential to grow into a real wide-spread implementation, dealing with assumptions and concerns which limit the current system. The standardization or adoption of such protocol would be a step towards public approval of electronic voting schemes, provided that the said protocol is secure and has been tested and tried.
  3. As indicated by the future work on these protocols, the applications for the proposed protocol are *not limited to government elections* only. These can be stretched to opinion polls or corporate elections providing a unified platform for voting regardless of the cost or circumstance.
  4. The release of new protocols, with *security issues* does not take steps to progress in public approval and, ultimately, replacement of the paper elections. A major effort has gone into development of a sound voting system and with the rapid developments of blockchain.

**10.BIBLIOGRAPHY**

**Websites**

1. <http://solidity.readthedocs.io/>
2. https://web3js.readthedocs.io
3. <http://blockchainhub.net/>
4. https://medium.com/
5. <https://tools.superdatascience.com/>
6. <https://hackernoon.com/>
7. <https://stackoverflow.com/>
8. https://blockgeeks.com/
9. https://www.bitdegree.org/learn/solidity-function-modifiers/

**Books**

1. Imran Bashir, ”Mastering Blockchain”, Edition 1, Packt Publishing, Birmingham, 2017
2. Mayukh Mukhopadhyay, “Ethereum Smart Contract Development”, Edition 1, Packt Publishing, Birmingham, 2018

**Appendix-I (Program Code)**

**A. Smart contracts for the system**

pragma solidity ^0.4.24;

contract voting{

mapping(bytes32 => uint8) public votesReceived;

bytes32[] public candidateList;

function voting(bytes32[] candidateName)

{

candidateList=candidateName;

}

function totalvotesfor(bytes32 candidate) view public returns(uint8)

{

require(validCandidate(candidate));

return votesReceived[candidate];

}

function voteforcandidate(bytes32 candidate) public

{

require(validCandidate(candidate));

votesReceived[candidate]+=1;

}

function validCandidate(bytes32 candidate) view public returns (bool)

{ for(uint i = 0; i <candidateList.length;i++)

{ if (candidateList[i]==candidate)

{ return true;

}

}

return false;

}

function getCandidateList() view public returns(bytes32[])

{ return candidateList;

}

function getWinner() view public returns(bytes32)

{

uint8 max=0;

for(uint8 i=0;i<candidateList.length;i++)

{

if(votesReceived[candidateList[i]]>max)

{ max=i;

}

}

return candidateList[max];

}

}

**B. React Front end**

import React, { Component } from 'react';

import './App.css';

import web3 from './web3';

import voting from './voting';

import hex2string from './hextostring.js';

const initCandidate = [ "0x434f4e4752455353000000000000000000000000000000000000000000000000", //congress "0x4248415254495941204a414e5441205041525459000000000000000000000000",//bjp "0x41414d204141444d492050415254590000000000000000000000000000000000", //aap

"0x4e4f544100000000000000000000000000000000000000000000000000000000" //nota

];

class App extends Component {

state = {

winner: '',

message: '',

vote: '',

votesfor1: '',

votesfor2: '',

votesfor3: '',

votesfor4: ''

};

async componentDidMount() {

console.log(voting.options.address);

const votesfor1 = await voting.methods.totalvotesfor(initCandidate[0]).call();

const votesfor2 = await voting.methods.totalvotesfor(initCandidate[1]).call();

const votesfor3 = await voting.methods.totalvotesfor(initCandidate[2]).call();

const votesfor4 = await voting.methods.totalvotesfor(initCandidate[3]).call();

this.setState({votesfor1});

this.setState({votesfor2});

this.setState({votesfor3});

this.setState({votesfor4});

}

onSubmit = async event => {

event.preventDefault();

const accounts = await web3.eth.getAccounts();

console.log('account is: ',accounts[0]);

this.setState({ message: 'Waiting on transaction success...' });

if(this.state.vote==="CONGRESS")

{

await voting.methods.voteforcandidate(initCandidate[0]).send({

from: accounts[0],

gas: '1000000' });

this.setState({ message: 'Thank you for voting!' });

}

else

{

if(this.state.vote==="BJP")

{

await voting.methods.voteforcandidate(initCandidate[1]).send({

from: accounts[0],

gas: '1000000' });

this.setState({ message: 'Thank you for voting!' });

}

else

{

if(this.state.vote==="AAP")

{

await voting.methods.voteforcandidate(initCandidate[2]).send({

from: accounts[0],

gas: '1000000' });

this.setState({ message: 'Thank you for voting!' });

}

else

{

if(this.state.vote==="NOTA")

{

await voting.methods.voteforcandidate(initCandidate[3]).send({

from: accounts[0],

gas: '1000000' });

this.setState({ message: 'Thank you for voting!' });

}

else

{

this.setState({ message: 'INVALID ENTRY' });

}

}

}

}

};

onClick = async () => {

const accounts = await web3.eth.getAccounts();

const winner1 = await voting.methods.getWinner().call();

const winner = hex2string(winner1);

this.setState({winner});

this.setState({votesfor1: ''});

this.setState({votesfor2: ''});

this.setState({votesfor3: ''});

this.setState({votesfor4: ''});

};

render() {

return (

<div className='header' >

<h2 id="headline">Decentralized Voting App</h2>

<p id="text">

There are currently{' '}

{3} candidates/political parties entered.

</p>

<hr />

<div className='column'>

<img src="http://www.trueopinion.co.in/wp-content/uploads/2017/03/27326e62bf6e213093d5b65c9b5efd61\_indian-national-congress-logo-congress-logo-clip-art\_992-1600.jpeg"

style={{ width: 400 , height: 350, float: 'left', top: this.props.top, left: this.props.left}}

alt-text="congress"

/>

<b>CONGRESS</b>

</div>

<div className='column' >

<img src="https://upload.wikimedia.org/wikipedia/commons/4/44/Bharatiya\_Janata\_Party\_%28icon%29.jpg"

style={{ width: 400, height: 350, float: 'left', top: this.props.top, left: this.props.left}}

alt-text="bjp"/>

<b font-size="1px">BJP</b>

</div>

<div className='column' >

<img src="https://www.hindustantimes.com/rf/image\_size\_640x362/HT/p1/2014/02/27/Incoming/Pictures/1188878\_Wallpaper2.jpg"

style={{width: 400, height: 350, float: 'left', top: this.props.top, left: this.props.left}}

alt-text="aap" />

<b>AAM AADMI PARTY</b>

</div>

<br />

<form onSubmit={this.onSubmit} style={{paddingTop:'450px'}} >

<h4 style={{color:'red'}}>Type the Political Party's Name</h4>

<div>

<input className="inpu"

value={this.state.vote}

onChange={ event => this.setState({ vote: event.target.value })}

/>

</div>

</form>

<h2>{this.state.message}</h2>

<hr />

<h2>CONGRESS: {this.state.votesfor1} </h2>

<hr />

<h2>BHARTIYA JANTA PARTY: {this.state.votesfor2} </h2>

<hr />

<h2>AAM AADMI PARTY: {this.state.votesfor3} </h2>

<hr />

<h2>NONE OF THE ABOVE(NOTA): {this.state.votesfor4} </h2>

<hr />

<button className="Button" onClick={this.onClick}><b>END VOTING</b></button>

<h2>The winner: {this.state.winner}</h2>

</div>

);

}

}

export default App;

**C. Node deploy script**

const HDWalletProvider = require('truffle-hdwallet-provider');

const Web3 = require('web3');

const { interface, bytecode } = require('./compile.js');

const provider = new HDWalletProvider(

'your 12 word mnemonic',

'your infura/rinkeby link'

);

const web3 = new Web3(provider);

const hex2str = require('./hextostring.js');

var initCandidate = ["0x434f4e4752455353000000000000000000000000000000000000000000000000",//congress

"0x4248415254495941204a414e5441205041525459000000000000000000000000",//bjp

"0x41414d204141444d492050415254590000000000000000000000000000000000"//aap];

console.log('initializing..');

//write a function JUST to use async ( bcoz there are

//two asynchron. calls in this program)

const deploy = async() => {

//get a list of all accounts

const accounts = await web3.eth.getAccounts();

//use one of those accounts to deploy contract

console.log('Attempting to deploy from account', accounts[0]);

const contractInstance = await new web3.eth.Contract(JSON.parse(interface))

.deploy({ data: bytecode, arguments: [initCandidate] })

.send({ from: accounts[0], gas: '1000000' });

//contractInstance.setProvider(provider);

console.log('Contract deployed to', contractInstance.options.address);

await contractInstance.methods.voteforcandidate(initCandidate[0]).send({ from: accounts[0], gas: '1000000' });

await contractInstance.methods.voteforcandidate(initCandidate[1]).send({ from: accounts[0], gas: '1000000' });

await contractInstance.methods.voteforcandidate(initCandidate[1]).send({ from: accounts[0], gas: '1000000' });

const cand1 = await contractInstance.methods.totalvotesfor(initCandidate[0]).call();

const cand2 = await contractInstance.methods.totalvotesfor(initCandidate[1]).call();

const cand3 = await contractInstance.methods.totalvotesfor(initCandidate[2]).call();

const winner = await contractInstance.methods.getWinner().call();

console.log('Votes for CONGRESS: '+ cand1);

console.log('\nVotes for BHARTIYA JANTA PARTY: '+ cand2);

console.log('\nVotes for AAM AADMI PARTY: '+ cand3);

console.log('\nThe Winner is: '+hex2str(winner));

};

deploy();

**Appendix-II (Sample Output/ Screenshot)**

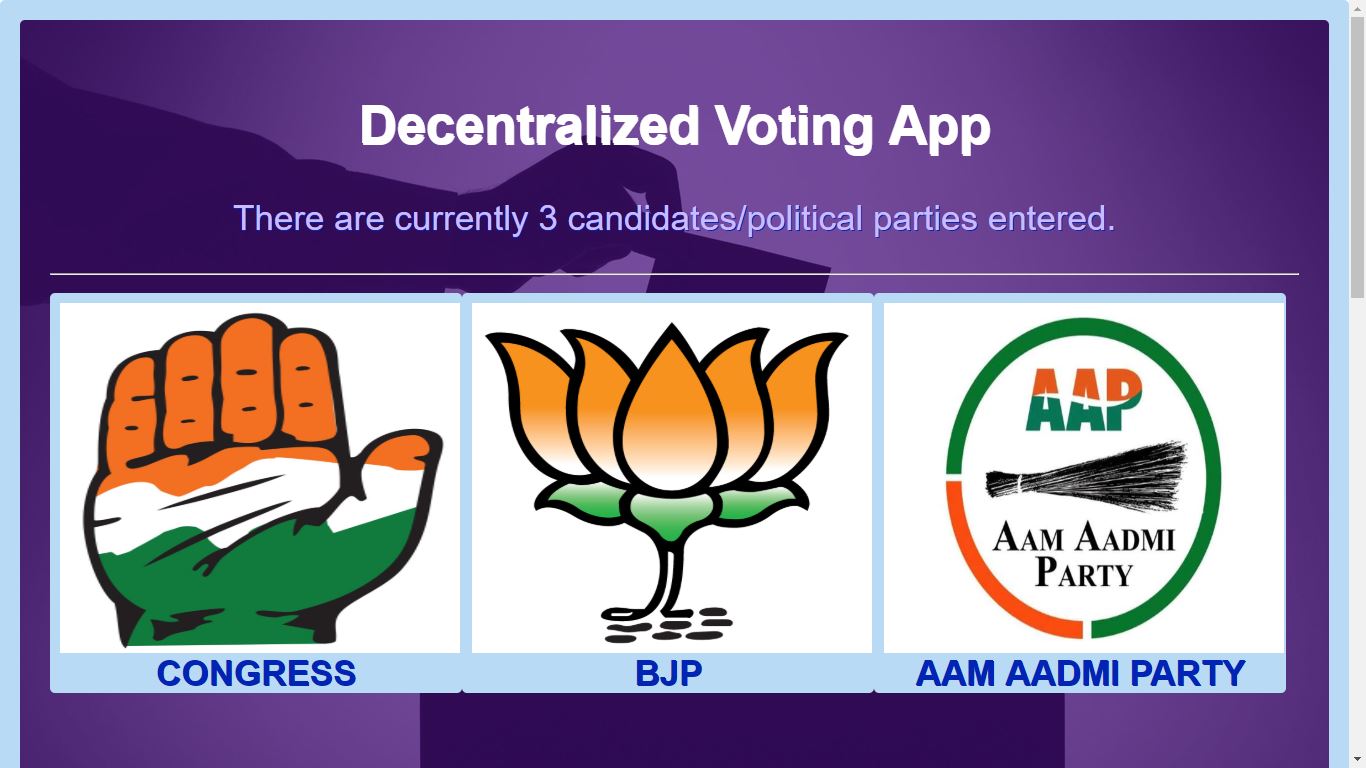
Fig.7: Main page

Fig.8: Voting a Political Party

Fig.9: After transaction is successful

Fig.10: Final result after all the votes are voted.