D-Vote: Decentralized Voting System Using Ethereum Blockchain

Darrell Yonathan
Department of Electrical Engineering
Faculty of Engineering
Universitas Indonesia
darrell.yonathan@ui.ac.id

Galih Damar Jati
Department of Electrical Engineering
Faculty of Engineering
Universitas Indonesia
galih.damar71@ui.ac.id

Geraldy Christanto
Department of Electrical Engineering
Faculty of Engineering
Universitas Indonesia
geraldy.christanto@ui.ac.id

Kenya Damayanti
Department of Electrical Engineering
Faculty of Engineering
Universitas Indonesia
kenyadamayantip@gmail.com

Jauzak Hussaini Windiatmaja
Department of Electrical Engineering
Faculty of Engineering
Universitas Indonesia
jauzak.hussaini@ui.ac.id

Riri Fitri Sari
Department of Electrical Engineering
Faculty of Engineering
Universitas Indonesia
riri@ui.ac.id

***Abstract*— Voting is a means of making collective decisions in communities, organizations, and institutions. Each participant is granted a number of votes that they can use to affect the election outcomes of the current candidates. The votes are tallied at the end of the voting process, and the outcome is determined by the electoral system in use. Conventionally, we may associate this voting mechanism with a piece of paper that is filled out and then placed in a box. However, postal ballots, postal voting, internet voting, and manual voting are all means of voting. The presence of incidents in which data from voting results are manipulated or are not counted from the indicated voting system calls the data's integrity into doubt. We think that utilizing blockchain technology for voting systems will overcome this problem. Blockchain will help secure voting result data and the voting process. A web-based application named D-Vote was developed to implement blockchain technology on voting systems. With blockchain characteristics, D-Vote can maintain the security of collected voting data as well as data containing the identities of voters who are entitled to vote. Furthermore, test result shows that all functions was successfully performed by entering several parameters as needed.**

Keywords—blockchain, decentralized, web-based application, security, voting

I. Introduction

Voting is a method of collective decision-making between communities, organizations and institutions. Voting is a widely used activity among the communities. Each participant is given a certain number of votes that they can cast to influence the election results of the existing candidates. At the end of the vote, then simultaneously the vote is counted and the result of the vote is determined according to the applicable electoral system. At first we may know this voting system through a paper that is filled and then collected into a place. However, voting can be done by several methods such as postal ballots, postal voting, online voting and machine voting (offline).

The existence of cases or events that data from the voting results are rigged or there is data that does not enter the conventional voting system or that is already online makes the integrity of the data questionable. This can be solved by the existence of a blockchain system. Blockchain will help secure voting result data and even secure the voting process. Blockchain has a decentralized mechanism so that every user in the voting can see the transaction data in its entirety. The verification process in this blockchain system also plays an important role to ensure that the people who votes are the ones who are indeed registered. This process will make the voting process much safer.

D-Vote is a web-based application designed to implement blockchain technology on campus voting systems. With the features of the blockchain, the application can easily authenticate the identity of the voter to prevent fraud in the vote and can prevent unregistered voters from voting. In addition, with blockchain technology, D-Vote can maintain the security of collected voting data as well as data containing the identities of voters who are entitled to vote.

The purpose of this research is to implement blockchain technology in voting systems using Ethereum. By using blockchain technology, the pre-recorded election results are secured, it cannot be changed or deleted. Decentralized systems in blockchain enhance user privacy as no third party can manipulate the data. In addition, voter identity is also more secure because it is pseudonymous, where users in the Ethereum network use private keys, so that others cannot know who is voting.

II. Literature Review

A. *Ethereum Blockchain*

As blockchain technology evolves, developers want to use the concept of blockchain into a wider range of applications not just as payment applications [1]. The protocol used in bitcoin is not able to accommodate these requirements, regardless of the advantages it has. So Vitalik Buterin developed Ethereum in 2013, as a platform that can create decentralized applications using blockchain through smart contract support [2].

Ethereum is an alternative open source platform for running decentralized applications by providing a very useful approach to various cases with fast development times and more efficient interactions [3]. In his paper, Buterin states that Ethereum allows developers to create blockchain-based applications, with features such as scalability, standardization, feature completeness, as well as ease of development and interoperability [4].

The networks in Ethereum can be distinguished into two types, namely public networks and private networks. Public networks can be accessed by anyone connected to the internet. On this network, anyone can view or make transactions on the blockchain network and validate the executed transactions [5]. Transaction approval is determined through the network node consensus mechanism. There are two kinds of public networks on Ethereum, namely the mainnet network which is the main network that has the actual coin value on the transactions made and the testnet network used for development before smart contracts are deployed to the production environment or to the mainnet [6].

A transaction is an instruction digitally signed by Externally-owned Account (EOA). EOA will initiate a transaction to update the status in the Ethereum network. With the decentralized nature of blockchain, new transactions must be distributed broadcast to the entire network [7]. Transactions require costs and must be validated through consensus protocols. Transaction fees in Ethereum using gas units.

B. Solidity

Solidity is a high-level programming language better known as contract-based. The syntax of this programming language is similar to that of the Javascript programming language. This programming language is used to improve the performance of virtual machines on Ethereum [8]. This solidity is a statically crafted scripting language for the verification process and limitations on compile time. In addition to compiling, this programming language will also help check at run-time. This solidity programming language also supports object-oriented programming such as object inheritance and others [9]. Solidity workflow is shown in Fig. 1.

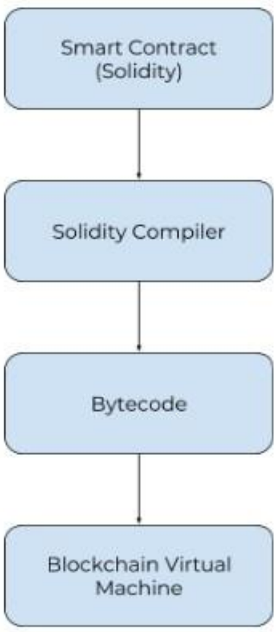


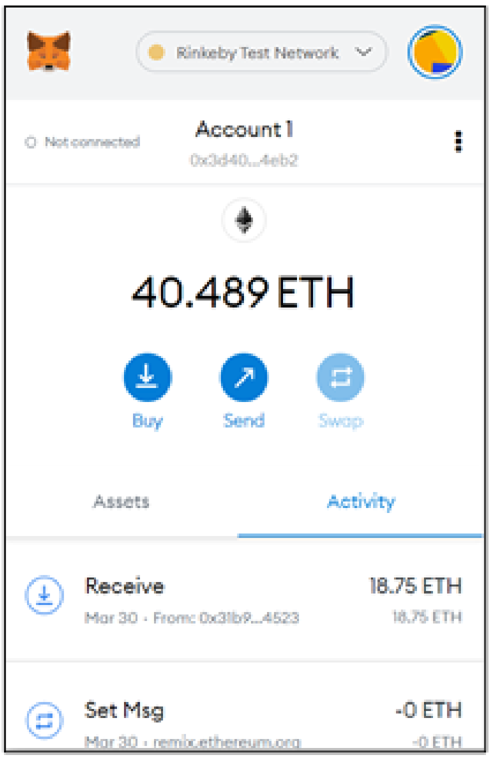
Fig. 1. Solidity Workflow

Solidity is a programming language created specifically for Smart Contracts on Ethereum [16]. Solidity is written in the .sol file format. Solidity is basically not an executable language on the Blockchain Virtual Machine [10]. Solidity is a language that aims to make it easier for developers to create smart contracts. When we are going to compile or deploy our smart contract, we need a solidity compiler [11]. Through the solidity compiler smart contract we will be compiled into bytecode which will be the bytecode that will be executed by the virtual machine.

C. Third Party Resources

Metamask is a cryptocurrency wallet that can be used in Chrome, Firefox and Brave browsers. This wallet application is a browser extension that works like a bridge between a normal browser and the Ethereum blockchain so that this application can also be referred to as a 3rd party application. Metamask can be used to store keys only for Ethereum cryptocurrencies [12]. There is an Ethereum cryptocurrency commonly called Ether. In addition there are also other cryptocurrencies built on the Ethereum blockchain that are tokens. Most of the tokens built on the Ethereum blockchain are called ERC20 tokens because they follow the rules that Ethereum developers have set to create new cryptocurrencies [13].

Metamask has several network options such as Rinkeby, Ropsten and other networks. This network serves to store transaction data from users. Metamask can interact automatically with Ethereum-based websites [14]. This metamask is quite safe because there hasn't been a major hack. Metamask browser extensions interface is shown in Fig. 2.



40.489 ETH

Receive 18.75 ETH
Mar 30 · From: 0x31b9...4523 18.75 ETH

Set Msg -0 ETH
Mar 30 · remix.ethereum.org -0 ETH

Fig. 2. Metamask Extension

Metamask is suitable for use as a network in application development commonly called Decentralize Application (DApp) [15]. Automation can also be done from DApp applications to Metamask to run and verify to blockchain networks [16]. Since this metamask is a wallet, users can also receive and send cryptocurrencies to other users. In metamask, users can also create more than one wallet account.

III. Proposed Work and Implementation

This section describes the proposed architecture and the implementation of D-Vote application. D-Vote is created based on Ethereum network architecture. D-Vote is connected using the Web3.js library. The smart contracts that we created are deployed into the Ethereum network through Ganache. We also use several accounts available in Ganache so that voters do not have to pay gas fee when voting.

A. *Directory*

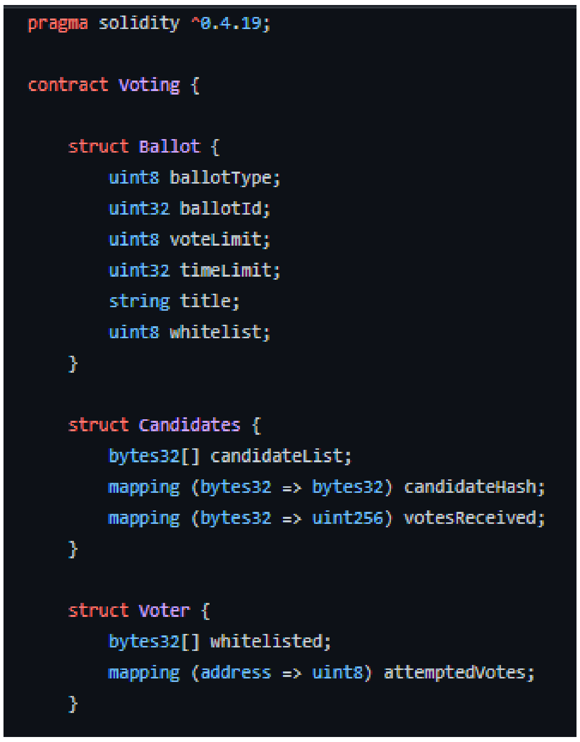
Programs are created using JavaScript with nodes.js framework. There are also several dependencies that need to be installed in this program. Front-end and back-end applications are stored in the app folder, while contracts are stored in the contracts folder. Contract migration scripts to other networks and initialization is done in the migrations folder.

B. *Contract (Solidity)*

There are three main contracts in our application, i.e. voting.sol, registrar.sol, and creator.sol. The diagram of each contract produces a different struct. The code snippet of the contracts are described as follow:

1) Voting.sol

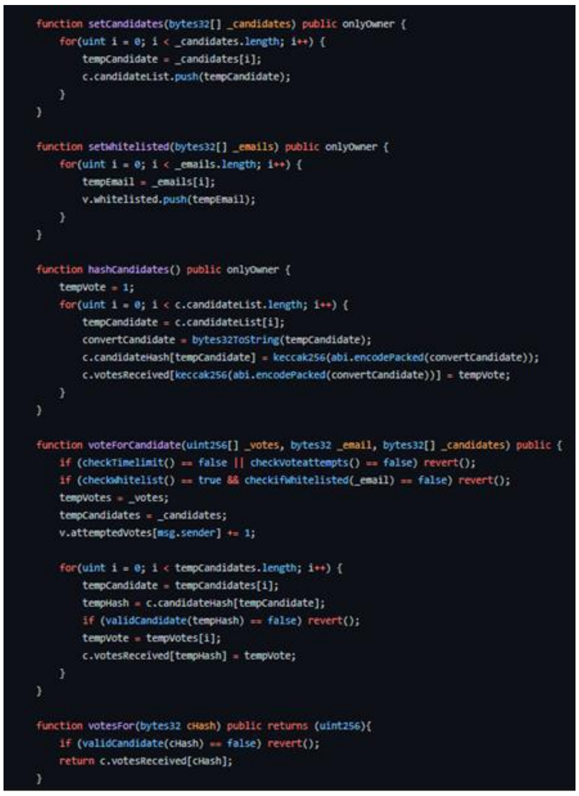
In this contract there are three data struct, i.e. Ballot as voting made, Candidates as data for the list of candidates on the ballot created, and Voters created using whitelist data or not. Votting.sol is shown in Fig. 3.



```
pragma solidity ^0.4.19;

contract Voting {

    struct Ballot {
        uint8 ballotType;
        uint32 ballotId;
        uint8 voteLimit;
        uint32 timeLimit;
        string title;
        uint8 whitelist;
    }

    struct Candidates {
        bytes32[] candidateList;
        mapping (bytes32 => bytes32) candidateHash;
        mapping (bytes32 => uint256) votesReceived;
    }

    struct Voter {
        bytes32[] whitelisted;
        mapping (address => uint8) attemptedVotes;
    }
```

Fig. 3. Voting.sol Snippet

To add the contents of the data to the struct, there are several functions that can be used. Function setCandidates is created for adding candidate data in ballots. Function hashCandidates is created for changing candidate names into Keccak256 format when saved into blockchain network. Function voteForCandidate is created to add the number of votes from voter to selected candidate, as well as some other additional functions such as to view candidate list or other data structure [16]. Candidate List and Voter Whitelist is shown in Fig. 4.

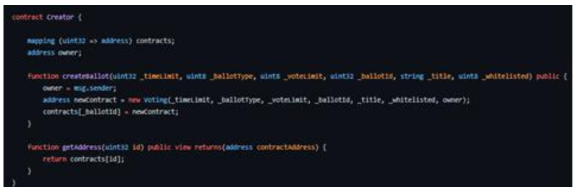


```
function setCandidates(bytes32[] _candidates) public onlyOwner {
    for(uint i = 0; i < _candidates.length; i++) {
        tempCandidate = _candidates[i];
        c.candidateList.push(tempCandidate);
    }
}

function setWhitelisted(bytes32[] _emails) public onlyOwner {
    for(uint i = 0; i < _emails.length; i++) {
        tempEmail = _emails[i];
        v.whitelisted.push(tempEmail);
    }
}

function hashCandidates() public onlyOwner {
    tempVote = 1;
    for(uint i = 0; i < c.candidateList.length; i++) {
        tempCandidate = c.candidateList[i];
        convertCandidate = bytes32ToString(tempCandidate);
        c.candidateHash[tempCandidate] = keccak256(abi.encodePacked(convertCandidate));
        c.votesReceived[keccak256(abi.encodePacked(convertCandidate))] = tempVote;
    }
}

function voteForCandidate(uint256[] _votes, bytes32 _email, bytes32[] _candidates) public {
    if (checkTimelimit() == false || checkVoteattempts() == false) revert();
    if (checkWhitelist() == true && checkifWhitelisted(_email) == false) revert();
    tempVotes = _votes;
    tempCandidates = _candidates;
    v.attemptedVotes[msg.sender] += 1;

    for(uint i = 0; i < tempCandidates.length; i++) {
        tempCandidate = tempCandidates[i];
        tempHash = c.candidateHash[tempCandidate];
        if (validCandidate(tempHash) == false) revert();
        tempVote = tempVotes[i];
        c.votesReceived[tempHash] = tempVote;
    }
}

function votesFor(bytes32 cHash) public returns (uint256){
    if (validCandidate(cHash) == false) revert();
    return c.votesReceived[cHash];
}
```

Fig. 4. Candidate List Snippets and Voter Whitelist

2) Creator.sol

Contract creator.sol is only used to create new ballots with some parameters such as, voting expiration time, ballot name, and whitelisted voter data if needed. Creator.sol snippet is shown in Fig. 5.

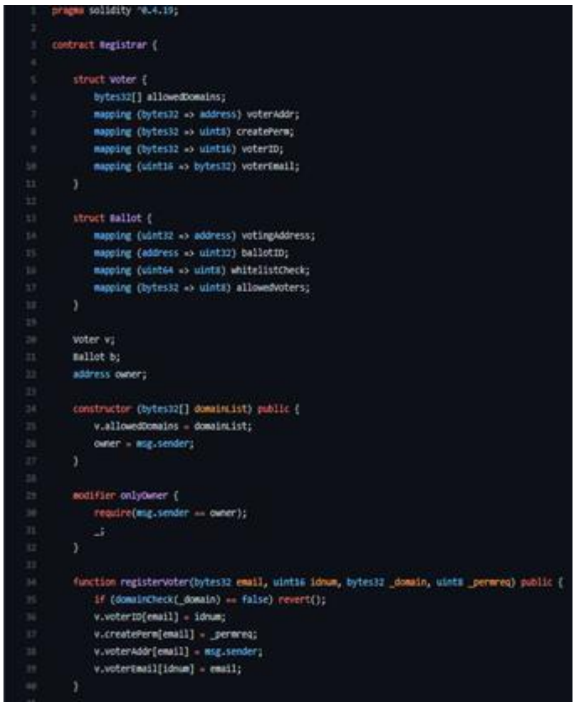


```
contract Creator {

    mapping (uint32 => address) contracts;
    address owner;

    function createBallot(uint32 _timeLimit, uint8 _ballotType, uint8 _voteLimit, uint32 _ballotId, string _title, uint8 _whitelisted) public {
        owner = msg.sender;
        address newContract = new Voting(_timeLimit, _ballotType, _voteLimit, _ballotId, _title, _whitelisted, owner);
        contracts[_ballotId] = newContract;
    }

    function getAddress(uint32 id) public view returns(address contractAddress) {
        return contracts[id];
    }
}
```

Fig. 5. Creator.sol Snippet

3) Registrar.sol

The function of the contract registrar is to register voters or voters using the appropriate email, as well as view the registration or registration of election participants. Registrar.sol is shown in Fig. 6.



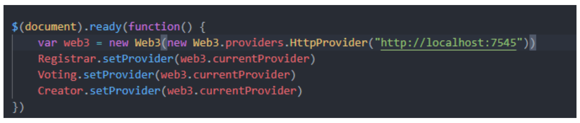
```
pragma solidity ^0.4.19;

contract Registrar {

    struct Voter {
        bytes32[] allowedDomains;
        mapping (bytes32 => address) voterAddr;
        mapping (bytes32 => uint8) createPerm;
        mapping (bytes32 => uint16) voterID;
        mapping (uint16 => bytes32) voterEmail;
    }

    struct Ballot {
        mapping (uint32 => address) votingAddress;
        mapping (address => uint32) ballotID;
        mapping (uint64 => uint8) whitelistCheck;
        mapping (bytes32 => uint8) allowedVoters;
    }

    Voter v;
    Ballot b;
    address owner;

    constructor (bytes32[] domainList) public {
        v.allowedDomains = domainList;
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    function registerVoter(bytes32 email, uint16 idnum, bytes32 _domain, uint8 _permreq) public {
        if (domainCheck(_domain) == false) revert();
        v.voterID[email] = idnum;
        v.createPerm[email] = _permreq;
        v.voterAddr[email] = msg.sender;
        v.voterEmail[idnum] = email;
    }
```

Fig. 6. Registrar.sol Snippet

The function of the contract registrar is to register voters or voters using the appropriate email, as well as view the registration or registration of election participants.

4) Web3 Initialization

Initialization of web3 instances in the app.js to be used in contract calling functions. Web3 is used to connect Ethereum wallets connected to the blockchain network. Contracts that have been compiled will be able to interact as objects in the created JavaScript. By using ganache, RPC Server is created in local network with port number 7545. Web3 initialization is shown in Fig. 7.

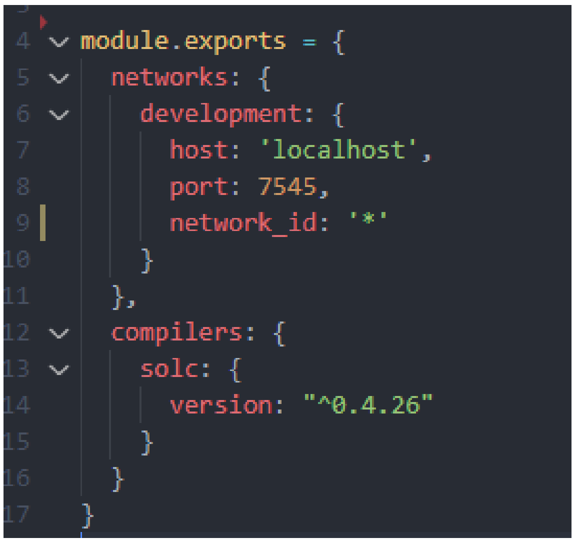


```
$(document).ready(function() {
    var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:7545"))
    Registrar.setProvider(web3.currentProvider)
    Voting.setProvider(web3.currentProvider)
    Creator.setProvider(web3.currentProvider)
})
```

Fig. 7. Web3 Initialization

5) Truffle Configuration

This file contains network and solc compiler configuration. Local network is used using Ganache so that transaction interactions conducted by users do not have to pay the required network fees. The solidity compiler version used in this research is 0.4.26. This compiler is adapted to the solidity version of the smart contract. The truffle configuration is shown in Fig. 8.



```
module.exports = {
  networks: {
    development: {
      host: 'localhost',
      port: 7545,
      network_id: '*'
    }
  },
  compilers: {
    solc: {
      version: "^0.4.26"
    }
  }
}
```

Fig. 8. Truffle Configuration

IV. Result and Analysis

This section described the result and analysis of the research. After the contract migration has been successfully performed from local network to Ganache network, we analyze application behavior and gas fee to make sure it is ready for production deployment. The analysis on Ganache Network is based on previous works [18]-[21].

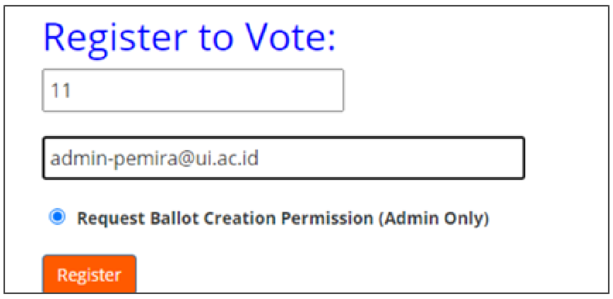
TABLE I. CONTRACT ADDRESS

Contract	Contract Address
Voting.sol	0xD8ACBcf0D96824a411A52c2B2bb75ddb33151DD9
Registrar.sol	0x48B52D9213d35BCF0Baa6A0a7C1b4A4dec1942b0
Creator.sol	0x6069e29859Eb6F419fE4Ef0391028b8f6Cf371E9

Tabel 1 described contract address of each contract deployed on the network. We test the application using defender.openzeppelin.com. Openzeppelin is used to speed up the testing process with an easier interface. Test result shows that all functions were successfully performed by entering several parameters as needed.

A. *Voter Registration*

Voter registration is done by entering an ID, i.e. student number, and also email address in form. Users can also select the option for request a permission so that the email can be used for admins in ballot creation. The voter registration interface is shown in Fig. 9.



Register to Vote:

11

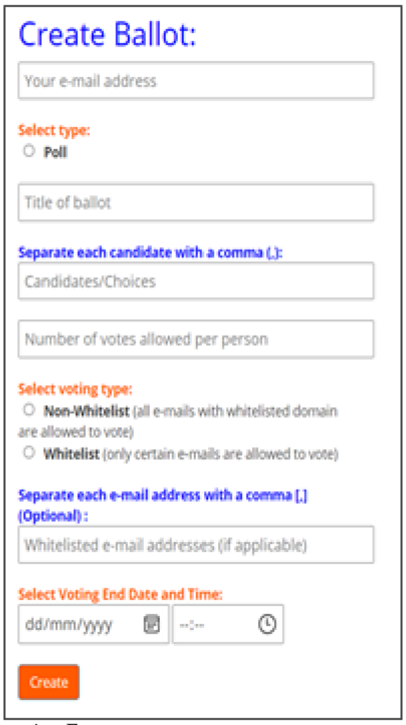
admin-pemira@ui.ac.id

● Request Ballot Creation Permission (Admin Only)

Fig. 9. Voter Registration

B. *Ballot Creation*

Ballot creation is done by filling out the following form. There are some information that needs to be provided in making the ballot. Users need to enter an email that has admin permissions. The type of ballot that can be created is poll. The poll will show the number of votes each ballot is reloaded. Users also need to add ballot titles, candidate names separated using commas, and limits on the number of votes allowed for each voter.



Create Ballot:

Your e-mail address

Select type:

○ Poll

Title of ballot

Separate each candidate with a comma (,):

Candidates/Choices

Number of votes allowed per person

Select voting type:

○ Non-Whitelist (all e-mails with whitelisted domain are allowed to vote)

○ Whitelist (only certain e-mails are allowed to vote)

Separate each e-mail address with a comma [,] (Optional) :

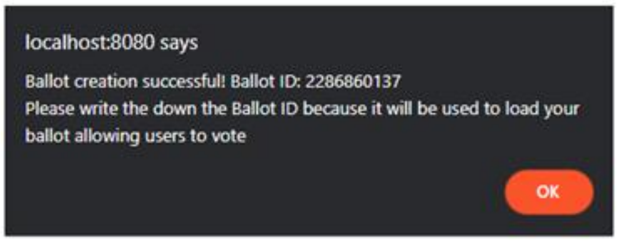
Whitelisted e-mail addresses (if applicable)

Select Voting End Date and Time:

dd/mm/yyyy --:--

Fig. 10. Ballot Creation Form

To vote, there are two options, i.e. non-whitelist and whitelist. The non-whitelist option allows all emails with whitelisted domains. For testing purposes, the whitelisted domain is @ui.ac.id as the project is intended for campus-level voting. The whitelist option only allows some emails to be populated in the next field. As in the candidate filling section, the filling of email addresses is separated by commas. Furthermore, the duration of voting is determined by entering the closing date and time of the voting period. After the ballot is successfully created, transaction information will appear containing address, ETH value, gas, and tx data. In addition, there are information about the type of contract created along with the function called and the parameters and inputs entered.



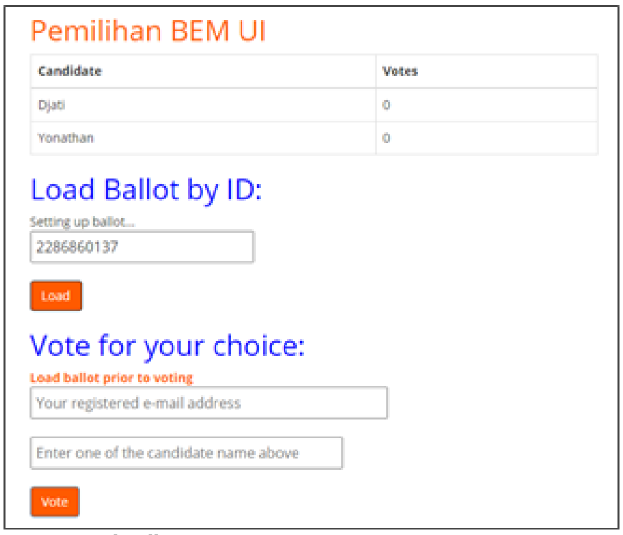
localhost:8080 says

Ballot creation successful! Ballot ID: 2286860137
Please write the down the Ballot ID because it will be used to load your ballot allowing users to vote

Fig. 11. Notification After Create Ballot Successful

C. *Load Ballot and Voting*

To vote, a ballot needs to be loaded first by entering its ID. After a ballot is successfully loaded, a table will appear containing the names of candidates along with the number of votes for each candidate.



Pemilihan BEM UI

Candidate	Votes
Djati	0
Yonathan	0

Load Ballot by ID:

Setting up ballot...

2286860137

Vote for your choice:

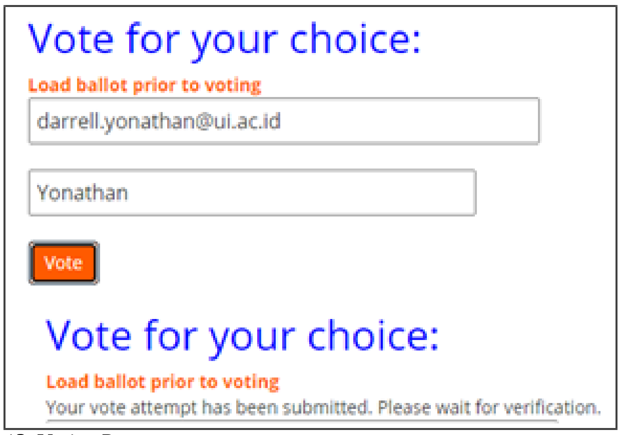
Load ballot prior to voting

Your registered e-mail address

Enter one of the candidate name above

Fig. 12. Load Ballot Process

To vote, the user needs to enter the email registered as the voter and also the name of the candidate you want to vote for.



Vote for your choice:

Load ballot prior to voting

darrell.yonathan@ui.ac.id

Yonathan

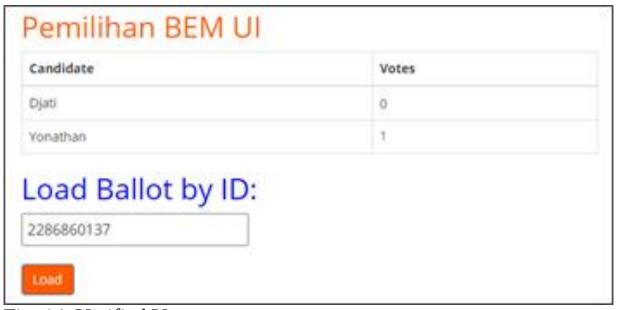
Vote for your choice:

Load ballot prior to voting

Your vote attempt has been submitted. Please wait for verification.

Fig. 13. Voting Page

A vote will be accepted if it has gone through the verification process. When verification is successful, a pop up like the one above appears. When the vote verification is successful, the number of votes on the selected candidate will increase if the ballot is loaded again.


Pemilihan BEM UI

Candidate	Votes
Djati	0
Yonathan	1

Load Ballot by ID:

2286860137

Fig. 14. Verified Vote

D. Gas Fee Analysis

The gas fee analysis is conducted to find out the average cost required for each transaction.

1) Contract Deployment

We can see from Table II that the total cost required to deploy the three contracts is 0.06409316 ETH. The most widely used cost is when deploying the contract creator because the contract is inheritance of voting contracts to be used at the time of creating a new ballot. For contracts that use the least cost is the contract registrar because on the contract the least computing.

TABLE II. ADDRESS CONTRACT

Contract	Gas Price (gwei)	Gas Used (unit)	Total Cost (ETH)
Voting	20	1173893	0.02347786
Registrar	20	599020	0.0119804
Creator	20	1431745	0.0286349

The most used cost is when deploying the contract creator. We believe that this might happened because the contract is inheritance of voting contracts to be used at the time of creating a new ballot. For contracts that use the least cost is the contract registrar because on the contract the least computing.

2) Transaction

Gas fee measurement in each transaction is done as many as 3 experiments to find the average cost required for each transaction:

TABLE III. TRANSACTION FEE FOR REGISTER NEW VOTER

n	Gas Used (unit)	Gas Price (gwei)	Tx Fee (ETH)
1	108246	20	0.00216492
2	108162	20	0.00216324
3	108066	20	0.00216132

We can see from Table III that the average transaction fee required to register a voter is 0.00216316 ETH.

TABLE IV. TRANSACTION FEE FOR FOR BALLOT MAKING

n	function	Gas Used (unit)	Gas Price (gwei)	Tx Fee (ETH)
1	checkVoter	1117325	20	0.00223465
	getPermission	63898	20	0.00127796
	createBallot	108554	20	0.00217108
	getAddress	181613	20	0.00363226
2	checkVoter	1117313	20	0.00223463
	getPermission	63886	20	0.00127772
	createBallot	132195	20	0.00026439
	getAddress	241666	20	0.00048333
3	checkVoter	1117325	20	0.00223465
	getPermission	63898	20	0.00012780
	createBallot	155836	20	0.00031167
	getAddress	301719	20	0.00060344

We can see from Table IV that the average transaction cost required for one ballot or voting is 0.005234542 ETH.

3) Vote

We can see from Table V that the average transaction fee made at the time of voting is 0.00635756 ETH. Over three transactions made in this voting system, the most costly transactions are voting. We believe it might happened because there are several functions needed in a voting contract. It also has a number of computational functions, such as hashing and encryption functions. The second transaction is at the time of ballot making where there are quite a lot of functions of various contracts called with varying gas costs. The last transaction that has the lowest transaction fee is voter registration, because of the little transaction data and few function calls made [17].

TABLE V. TRANSACTION FEE FOR VOTE

n	Gas Used (unit)	Gas Price (gwei)	Tx Fee (ETH)
1	316248	20	0.00632496
2	243903	20	0.00487806
3	393544	20	0.00787066

V. Conclusion and Future Work

A blockchain-based e-voting application used to conduct candidate voting is successfully made. Blockchain technology is used to maintain data security and integrity as well as to ensure voter identity to avoid fraud in the voting process. The immutable nature of the blockchain will also make the data of the voting process cannot be changed and always relate to each transaction. We aware that these applications still have some drawbacks, especially on the application interface that are less attractive to use. Further development and research of the user interface and user experience are needed. Authentication also needs to be added to the administrator when registering a voter and creating a new vote.

Acknowledgment

This work is supported by the Ministry of Education, Culture, Research, and Technology (Kemendikbudristek) of Indonesia under PDD Grant with contract number NKB-1016/UN2.RST/HKP.05.00/2022.

References

[1] Y. Huang, B. Wang and Y. Wang, "MResearch on Ethereum Private Blockchain Multi-nodes Platform," 2020 International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE), 2020, pp. 369-372, doi: 10.1109/ICBAIE49996.2020.00083.

[2] S. Rouhani and R. Deters, "Performance analysis of ethereum transactions in private blockchain," 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), 2017, pp. 70-74, doi: 10.1109/ICSESS.2017.8342866.

[3] X. Liu, R. Chen, Y. Chen and S. Yuan, "Off-chain Data Fetching Architecture for Ethereum Smart Contract," 2018 International Conference on Cloud Computing, Big Data and Blockchain (ICCBB), 2018, pp. 1-4, doi: 10.1109/ICCBB.2018.8756348.

[4] G. A. Pierro and H. Rocha, "The Influence Factors on Ethereum Transaction Fees," 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 2019, pp. 24-31, doi: 10.1109/WETSEB.2019.00010.

[5] C. BouSaba and E. Anderson, "Degree Validation Application Using Solidity and Ethereum Blockchain," 2019 SoutheastCon, 2019, pp. 15, doi: 10.1109/SoutheastCon42311.2019.9020503.

[6] S. H. Maeng, M. Essaid and H. T. Ju, "Analysis of Ethereum Network Properties and Behavior of Influential Nodes," 2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS), 2020, pp. 203-207, doi: 10.23919/APNOMS50412.2020.9236965.

[7] R. A. Canessane, N. Srinivasan, A. Beuria, A. Singh and B. M. Kumar, "Decentralised 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.

[8] Y. N. Aung and T. Tantidham, "Review of Ethereum: Smart home case study," 2017 2nd International Conference on Information Technology (INCIT), 2017, pp. 1-4, doi: 10.1109/INCIT.2017.8257877.

[9] P. V. Sukharev and D. S. Silnov, "Asynchronous Mining of Ethereum Cryptocurrency," 2018 IEEE International Conference "Quality Management, Transport and Information Security, Information Technologies" (IT&QM&IS), 2018, pp. 731-735, doi:10.1109/ITMQIS.2018.8524929.

[10] W. Chan and A. Olmsted, "Ethereum transaction graph analysis," 2017 12th International Conference for Internet Technology and Secured Transactions (ICITST), 2017, pp. 498-500, doi: 10.23919/ICITST.2017.8356459.

[11] P. Deshmukh, S. Kalwaghe, A. Appa and A. Pawar, "Decentralised Freelancing using Ethereum Blockchain," 2020 International Conference on Communication and Signal Processing (ICCSP), 2020, pp. 881-883, doi: 10.1109/ICCSP48568.2020.9182127.

[12] Aldweesh, M. Alharby and A. van Moorsel, "Performance Benchmarking for Ethereum Opcodes," 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA), 2018, pp. 1-2, doi: 10.1109/AICCSA.2018.8612882.

[13] V. Aleksieva, H. Valchanov and A. Huliyan, "Application of Smart Contracts based on Ethereum Blockchain for the Purpose of Insurance Services," 2019 International Conference on Biomedical Innovations and Applications (BIA), 2019, pp. 1-4, doi: 10.1109/BIA48344.2019.8967468

[14] J. Chen, "Finding Ethereum Smart Contracts Security Issues by Comparing History Versions," 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2020, pp. 1382-1384..

[15] P. V. Sukharev and D. S. Silnov, "Mining Result Validation in the Ethereum Cryptocurrency," 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2019, pp. 1731-1734, doi: 10.1109/EIConRus.2019.8656906.

[16] T. Wang, C. Zhao, Q. Yang, S. Zhang and S. C. Liew, "Ethna: Analyzing the Underlying Peer-to-Peer Network of Ethereum Blockchain," in IEEE Transactions on Network Science and Engineering, doi: 10.1109/TNSE.2021.3078181.

[17] U. Bagadia, J. Bodkurwar, J. Bhat and A. Halbe, "Performance Analysis of Decentralized Ethereum Blockchain System," 2020 International Conference on Inventive Computation Technologies (ICICT), 2020, pp. 127-131, doi: 10.1109/ICICT48043.2020.9112550.

[18] P. Hegedus, "Towards Analyzing the Complexity Landscape of Solidity Based Ethereum Smart Contracts," 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 2018, pp. 35-39.

[19] S. Kim, J. Song, S. Woo, Y. Kim and S. Park, "Gas Consumption Aware Dynamic Load Balancing in Ethereum Sharding Environments," 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W), 2019, pp. 188-193, doi: 10.1109/FAS-W.2019.00052.

[20] D. Son, S. Al Zahr and G. Memmi, "Performance Analysis of an Energy Trading Platform Using the Ethereum Blockchain," 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2021, pp. 1-3, doi: 10.1109/ICBC51069.2021.9461115.

[21] Aldweesh, "Analysis, Evaluation and Benchmark The Ethereum Incentive Mechanism," 2021 IEEE 11th International Conference on Electronics Information and Emergency Communication (ICEIEC)2021 IEEE 11th International Conference on Electronics Information and Emergency Communication (ICEIEC), 2021, pp. 14, doi: 10.1109/ICEIEC51955.2021.9463723.