**School of Computer Science and Engineering**

**Blockchain Technologies**

**Review – 3**

**Course Faculty:** Jothi K R **Date:** 2.oct.2023

| S.no | Team member's name | Reg.no |
|---|---|---|
| 1) | Suriya Narayanan | 20MID0059 |
| 2) | Guruarunachalam | 20MID0068 |
| 3) | Prabhath Sai | 20MID0137 |

# Decentralized Voting System with Ethereum in Blockchain using Solidity and Web3 Technology

## Introduction:

The project aims to revolutionize conventional voting systems by implementing a Decentralized Voting System leveraging Ethereum's blockchain, Solidity smart contracts, and Web3 technology. This endeavor responds to the critical need for transparent, secure, and accessible voting mechanisms in contemporary societies. By harnessing the inherent attributes of blockchain—decentralization, immutability, and transparency—the proposed system endeavors to mitigate common issues prevalent in centralized voting structures, such as fraud, tampering, and lack of verifiability. The integration of Solidity, Ethereum's programming language for smart contracts, enables the design of a robust voting logic. Additionally, Web3 technology facilitates user interaction with the decentralized application, ensuring a seamless and intuitive voting experience. This project not only endeavors to create a functional

decentralized voting system but also aims to contribute to the broader discourse on leveraging blockchain for democratic processes.

**Background and Literature Review :**

The background contextualizes the Decentralized Voting System's development within the landscape of blockchain technology, emphasizing Ethereum's significance in executing smart contracts. Blockchain's immutable ledger and decentralized structure form the foundation for transparent and secure voting mechanisms. Solidity, as Ethereum's primary programming language, is pivotal in defining the voting logic, ensuring the integrity of the process. Existing literature underscores the necessity of decentralized voting systems, highlighting challenges faced by traditional centralized methods, such as susceptibility to manipulation and lack of transparency. Research into similar projects showcases the potential of blockchain, Solidity, and Web3 technology in revolutionizing democratic processes. This review serves as a vital foundation, acknowledging the innovative potential of these technologies while considering insights and advancements made in decentralized voting systems to inform the development of the current project.

**Objectives**:

The objectives of this project encompass the development and implementation of a robust Decentralized Voting System using Ethereum's blockchain infrastructure, Solidity smart contracts, and Web3 technology. The primary goal is to create a transparent, secure, and tamper-resistant voting platform capable of ensuring the integrity of the voting process. Specifically, the system aims to leverage blockchain's decentralized nature to eliminate vulnerabilities associated with centralized voting systems, fostering trust and transparency among participants. The scope of this endeavor encompasses the design and deployment of smart contracts written in Solidity, integration with Web3 for user interaction, and the creation of a user-friendly interface. The project's focus remains on achieving a functional prototype that demonstrates the feasibility and potential of decentralized voting systems, with considerations for scalability and adaptability to real-world voting scenarios.

**Security and Trustworthiness:**

Through the following properties, blockchain technology improves voting system security and trust:

Immutable Ledger: Transactions are tamper-resistant once they are recorded on the blockchain, preventing fraud and data tampering.

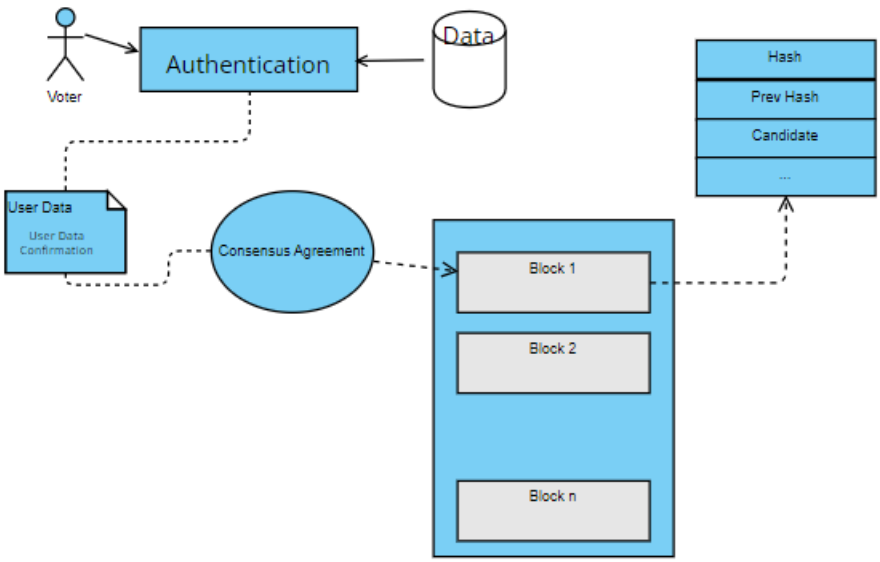
Cryptographic Verification: To confirm the validity of participants, cryptographic approaches verify voters and their votes.

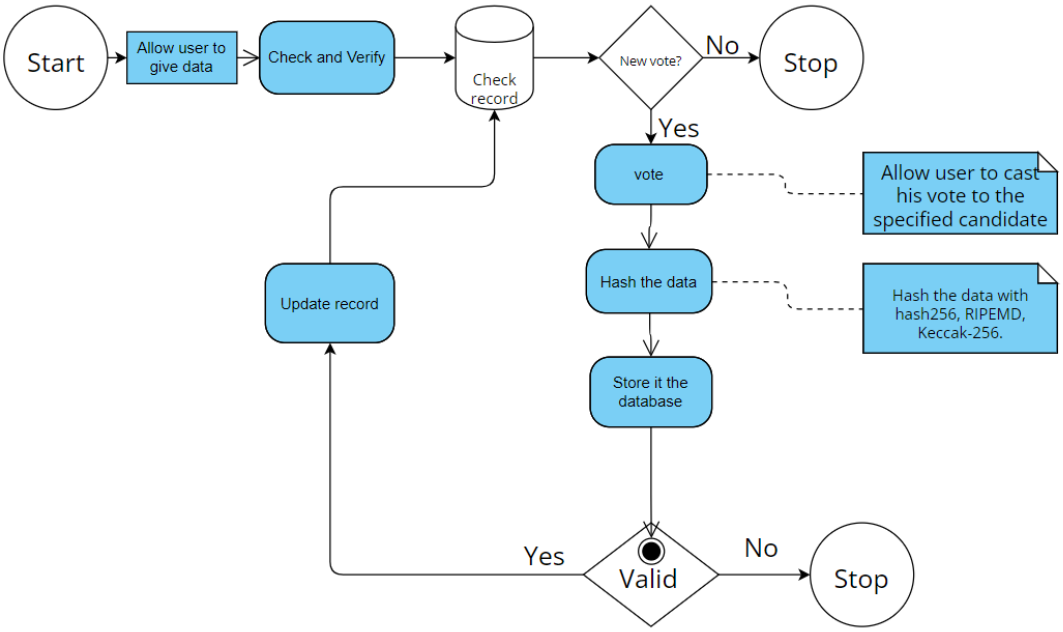
Decentralisation: The danger of unauthorised access and manipulation by a single entity is decreased by the lack of a central authority.

Transparency: All transactions are open to the public, enabling unbiased auditing of the voting procedure and outcomes.

Elimination of Double Voting: Blockchain's unique identification and consensus processes stop double voting, improving the election's integrity.

**System and Architecture:**






**Methodology:**

The methodology employed in developing the Decentralized Voting System involves a systematic approach integrating Ethereum, Solidity, and Web3 technologies. Initially, extensive research was conducted to understand blockchain fundamentals, Ethereum's smart contract capabilities, and Solidity's syntax and functionality. The development phase commenced with the creation of smart contracts using Solidity, defining the rules and logic governing the voting process. Concurrently, Web3 integration facilitated user interactions, enabling seamless access to the decentralized application (DApp). Continuous testing and refinement iteratively validated the system's functionality, including smart contract reliability, frontend

usability, and backend interactions. This iterative development cycle involved agile methodologies, allowing for flexibility and responsiveness to evolving requirements. The project emphasized a collaborative approach, leveraging best practices in blockchain development, and soliciting feedback from stakeholders to ensure the integrity and usability of the Decentralized Voting System.

**Implementation:**

The implementation phase of the Decentralized Voting System involved a structured and iterative approach, leveraging Ethereum's blockchain, Solidity smart contracts, and Web3 technology. The development process commenced with the creation of Solidity smart contracts, constituting the backbone of the voting system. These contracts encapsulated the essential voting logic, ensuring the integrity and transparency of the entire process. The smart contracts were meticulously designed to handle ballot creation, voter registration, vote casting, and tallying, employing cryptographic techniques to secure and validate transactions.

Simultaneously, the user interface (UI) was crafted to facilitate seamless user interactions via Web3-enabled decentralized applications. This front-end design focused on providing voters with an intuitive and accessible platform to cast their votes

securely. Web3 integration enabled smooth communication between the UI and the Ethereum blockchain, allowing voters to interact with the voting system transparently and securely.

The back-end development was dedicated to ensuring the robustness and efficiency of the system. This involved creating middleware and backend processes that interacted with the Ethereum network, managing the flow of data between the UI and the blockchain while upholding security measures.

Challenges during implementation included ensuring scalability, optimizing gas fees, and addressing potential security vulnerabilities. Strategies were employed to mitigate these challenges, including code optimization for efficient gas usage, rigorous testing for security loopholes, and modular design for scalability.

Continuous testing and debugging were integral parts of the implementation phase, with rigorous testing procedures conducted to validate the functionality and security of the entire system. Unit tests, integration tests, and simulated user interactions were employed to identify and rectify any discrepancies or vulnerabilities.

The iterative development approach allowed for constant improvements and refinements based on feedback and testing

results. Stakeholder involvement and peer review further ensured the system's reliability, usability, and adherence to the project's objectives. Overall, the implementation phase was characterized by meticulous development, testing, and refinement to create a robust and functional Decentralized Voting System.

**Code:**

- **Solidity code:**

```
//SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Voting {
    struct Candidate {
        string name;
        uint256 voteCount;
    }

    Candidate[] public candidates;

    mapping(address => bool) public hasVoted;
    mapping(address => string) public voterChoice;
    mapping(address => mapping(bytes32 => bool)) public
voteRecords;

    event Voted(address indexed voter, string indexed
candidate);
```

```
    constructor() {
        candidates.push(Candidate("Candidate 1", 0));
        candidates.push(Candidate("Candidate 2", 0));
        candidates.push(Candidate("Candidate 3", 0));
        candidates.push(Candidate("Candidate 4", 0));
    }

    function vote(string memory candidateName) public
returns (string memory,bytes32) {
        bool validCandidate = false;

        for (uint256 i = 0; i < candidates.length; i++) {
            if (keccak256(bytes(candidates[i].name)) ==
keccak256(bytes(candidateName))) {
                validCandidate = true;
                break;
            }
        }

        require(validCandidate, "Invalid candidate name");
        require(!hasVoted[msg.sender], "You have already
voted");



        bytes32 h1 = keccak256(abi.encodePacked(msg.sender,
candidateName));
        voteRecords[msg.sender][h1] = true;

        for (uint256 i = 0; i < candidates.length; i++) {
```

```
            if (keccak256(bytes(candidates[i].name)) ==
keccak256(bytes(candidateName))) {
                candidates[i].voteCount++;
                hasVoted[msg.sender] = true;
                voterChoice[msg.sender] = candidateName;
                emit Voted(msg.sender, candidateName);
                return ("your voted unquie address: ",h1);
            }
        }
        return ("your voted unquie address: ",h1);
    }

    function getVoteCount(string memory candidateName)
public view returns (uint256) {
        for (uint256 i = 0; i < candidates.length; i++) {
            if (keccak256(bytes(candidates[i].name)) ==
keccak256(bytes(candidateName))) {
                return candidates[i].voteCount;
            }
        }
        revert("Invalid candidate name");
    }

    function getCandidateCount() public view returns
(uint256) {
        return candidates.length;
    }
```

```
    function checkVote(address userAddress, string memory
candidateName) public view returns (string memory,
bytes32,bytes32) {
        bytes32 h2 =
keccak256(abi.encodePacked(userAddress,
candidateName));

        bytes32 h1 =
keccak256(abi.encodePacked(userAddress,
voterChoice[userAddress]));

        if (voteRecords[userAddress][h1] && h1 == h2) {
            return ("Vote is correctly casted",h1,h2);
        } else {
            return ("Vote is not correctly casted or check your
input",h1,h2);
        }
    }
}
```

- **Html code:**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Decentralized Voting System</title>
  <!-- Bootstrap CSS -->
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/c
ss/bootstrap.min.css">
```

```
</head>
<body>
  <div class="container mt-4">
    <h1>Decentralized Voting System</h1>
    <div class="form-group mt-4">
      <label for="voterAddress">Voter's Address:</label>
      <input type="text" class="form-control"
id="voterAddress">
    </div>
    <div class="form-group">
      <label for="candidateName">Candidate Name:</label>
      <input type="text" class="form-control"
id="candidateName">
    </div>
    <div class="form-group">
      <button onclick="castVote()" class="btn btn-
primary">Cast Vote</button>
    </div>
    <div class="form-group mt-4">
      <label for="voteHash">Vote Hash (h1):</label>
      <input type="text" class="form-control" id="voteHash"
readonly>
    </div>
    <div class="form-group">
      <label for="checkCandidateName">Candidate
Name:</label>
      <input type="text" class="form-control"
id="checkCandidateName">
    </div>
```

```
    <div class="form-group">
      <button onclick="checkVote()" class="btn btn-
primary">Check Vote</button>
    </div>
    <div class="form-group mt-4">
      <label for="candidateVoteCount">Candidate Vote
Count:</label>
      <input type="text" class="form-control"
id="candidateVoteCount">
    </div>
    <div class="form-group">
      <button onclick="getCandidateVoteCount()" class="btn
btn-primary">Get Vote Count</button>
    </div>
  </div>

  <!-- Bootstrap JS and Web3.js -->
  <script src="https://code.jquery.com/jquery-
3.5.1.slim.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/@metamask/detect-
provider/dist/detect-provider.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/web3@1.5.2/dist/web3.
min.js"></script>
  <script src="script.js"></script>
</body>
</html>
```

- **Web3 code:**

```
// Replace 'YourContractAddress' and 'YourContractABI'
with actual values
const contractAddress =
"0x8b358b97434b034B7839406D24D4D47780784448";
const contractABI = [
 {
  inputs: [],
  stateMutability: "nonpayable",
  type: "constructor",
 },
 {
  inputs: [
   {
    internalType: "string",
    name: "candidateName",
    type: "string",
   },
  ],
  name: "vote",
  outputs: [
   {
    internalType: "string",
    name: "",
    type: "string",
   },
   {
    internalType: "bytes32",
    name: "",
    type: "bytes32",
```

```
    },
  ],
  stateMutability: "nonpayable",
  type: "function",
},
{
  anonymous: false,
  inputs: [
    {
      indexed: true,
      internalType: "address",
      name: "voter",
      type: "address",
    },
    {
      indexed: true,
      internalType: "string",
      name: "candidate",
      type: "string",
    },
  ],
  name: "Voted",
  type: "event",
},
{
  inputs: [
    {
      internalType: "uint256",
      name: "",
```

```
      type: "uint256",
    },
  ],
  name: "candidates",
  outputs: [
    {
      internalType: "string",
      name: "name",
      type: "string",
    },
    {
      internalType: "uint256",
      name: "voteCount",
      type: "uint256",
    },
  ],
  stateMutability: "view",
  type: "function",
},
{
  inputs: [
    {
      internalType: "address",
      name: "userAddress",
      type: "address",
    },
    {
      internalType: "string",
      name: "candidateName",
```

```
      type: "string",
    },
  ],
  name: "checkVote",
  outputs: [
    {
      internalType: "string",
      name: "",
      type: "string",
    },
    {
      internalType: "bytes32",
      name: "",
      type: "bytes32",
    },
    {
      internalType: "bytes32",
      name: "",
      type: "bytes32",
    },
  ],
  stateMutability: "view",
  type: "function",
},
{
  inputs: [],
  name: "getCandidateCount",
  outputs: [
    {
```

```
      internalType: "uint256",
      name: "",
      type: "uint256",
    },
  ],
  stateMutability: "view",
  type: "function",
},
{
  inputs: [
    {
      internalType: "string",
      name: "candidateName",
      type: "string",
    },
  ],
  name: "getVoteCount",
  outputs: [
    {
      internalType: "uint256",
      name: "",
      type: "uint256",
    },
  ],
  stateMutability: "view",
  type: "function",
},
{
  inputs: [
```

```
    {
      internalType: "address",
      name: "",
      type: "address",
    },
  ],
  name: "hasVoted",
  outputs: [
    {
      internalType: "bool",
      name: "",
      type: "bool",
    },
  ],
  stateMutability: "view",
  type: "function",
},
{
  inputs: [
    {
      internalType: "address",
      name: "",
      type: "address",
    },
  ],
  name: "voterChoice",
  outputs: [
    {
      internalType: "string",
```

```
      name: "",
      type: "string",
    },
  ],
  stateMutability: "view",
  type: "function",
},
{
  inputs: [
    {
      internalType: "address",
      name: "",
      type: "address",
    },
    {
      internalType: "bytes32",
      name: "",
      type: "bytes32",
    },
  ],
  name: "voteRecords",
  outputs: [
    {
      internalType: "bool",
      name: "",
      type: "bool",
    },
  ],
  stateMutability: "view",
```

```
    type: "function",
  },
];

let web3;
let contractInstance;

// Function to initialize Web3
async function initWeb3() {
  if (window.ethereum) {
    web3 = new Web3(window.ethereum);
    try {
      await window.ethereum.enable();
      initContract();
    } catch (error) {
      console.error("User denied account access");
    }
  } else if (window.web3) {
    web3 = new Web3(web3.currentProvider);
    initContract();
  } else {
    console.error("No web3 detected. Please install
MetaMask");
  }
}

// Function to initialize Contract instance
function initContract() {
```

```
  contractInstance = new web3.eth.Contract(contractABI,
contractAddress);
}

// Function to cast vote
async function castVote() {
  const voterAddress =
document.getElementById("voterAddress").value;
  const candidateName =
document.getElementById("candidateName").value;

  try {
    const result = await contractInstance.methods
      .vote(candidateName)
      .send({ from: voterAddress });
    console.log(result);
    document.getElementById("voteHash").value =
      result.events.Voted.returnValues[1]; // Assuming h1 is
returned as the second parameter
  } catch (error) {
    console.error("Error casting vote:", error);
  }
}

// Function to check if the vote is correctly casted
async function checkVote() {
  const voterAddress =
document.getElementById("voterAddress").value;
  const checkCandidateName =
```

```
document.getElementById("checkCandidateName").value;

  try {
    const result = await contractInstance.methods
      .checkVote(voterAddress, checkCandidateName)
      .call();
    console.log(result);
    // Display the vote status or perform required action with
the result
  } catch (error) {
    console.error("Error checking vote:", error);
  }
}

// Function to get vote count for a candidate
async function getCandidateVoteCount() {
  const candidateName =
document.getElementById("candidateName").value;

  try {
    const result = await contractInstance.methods
      .getVoteCount(candidateName)
      .call();
    console.log(result);
    document.getElementById("candidateVoteCount").value
= result;
  } catch (error) {
    console.error("Error getting vote count:", error);
```

```
  }
}

// Initialize Web3 and Contract on window load
window.onload = function () {
  initWeb3();
};
```

**Case Studies:**

For the most recent information, use individual case studies or projects:

Follow My Vote: The Follow My Vote project used Ethereum to provide a safe and transparent online voting system. They recorded votes and created secure voter IDs using Ethereum's smart contracts.

Voatz: Although Hyperledger was utilised largely by Voatz, it is still important to note. To improve accessibility and security, they investigated mobile devices and blockchain-based voting.

Despite not being a member of Ethereum, Estonia is renowned for its innovative work in e-government. They looked at using blockchain for a variety of services, including electronic voting for shareholders in some businesses.

Kleros: Kleros is an Ethereum-based decentralised platform for dispute settlement. It demonstrates how Ethereum may be used for

decision-making inside a decentralised network, and not just for voting.

Aragon: Aragon is an Ethereum-based platform with an emphasis on governance. It enables businesses to design decentralised structures with open voting procedures.

**Challenges and Limitations:**

Although intriguing, implementing decentralised voting systems has a number of drawbacks:

Scalability: Ensuring that a blockchain can efficiently conduct a large-scale election with millions of votes is a difficult task.

Voter privacy: It might be challenging to protect voter privacy when recording votes on a public ledger. Technologies that improve privacy, such as zero-knowledge proofs (zk-SNARKs), may be able to aid.

Voter authentication: It can be difficult and lead to identity fraud to verify voters' identities in a decentralised fashion without depending on a central authority.

Accessibility: Not every voter will have internet access or the technological skills necessary to take part in a blockchain-based election.

Regulatory Compliance: It might be difficult to comply with legal and regulatory standards, such as confirming a voter's eligibility, because they differ between jurisdictions.

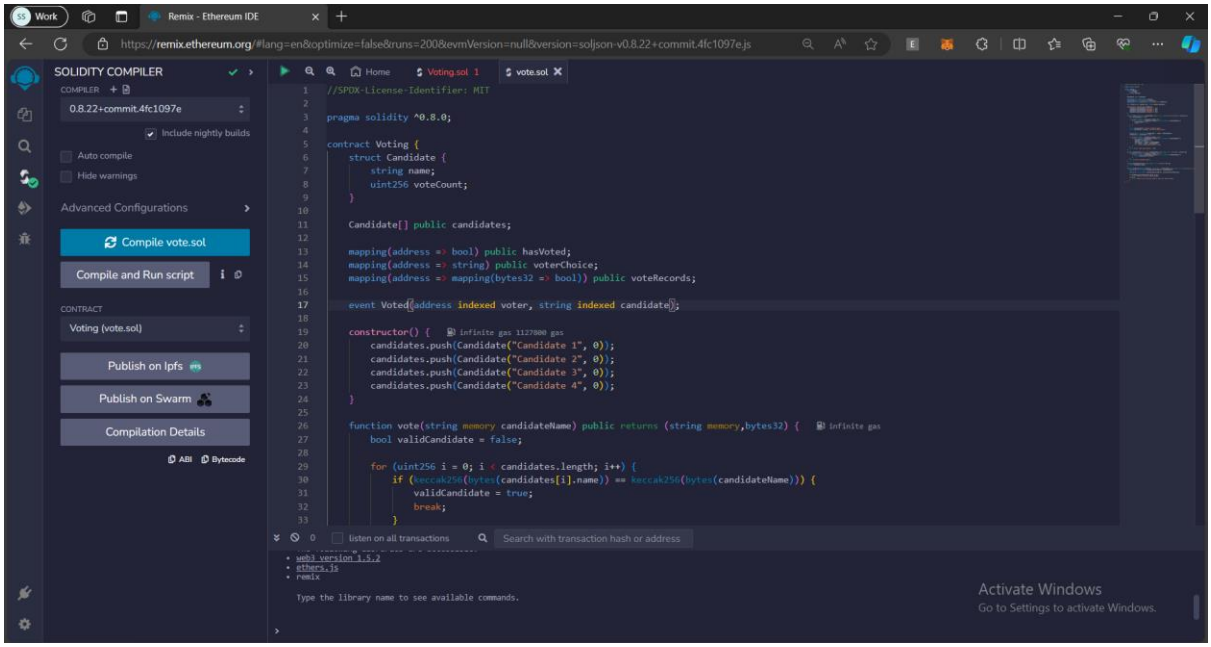
Security: Although blockchain is safe, smart contracts or the network layer may have flaws. To thwart assaults, robust security measures are necessary.

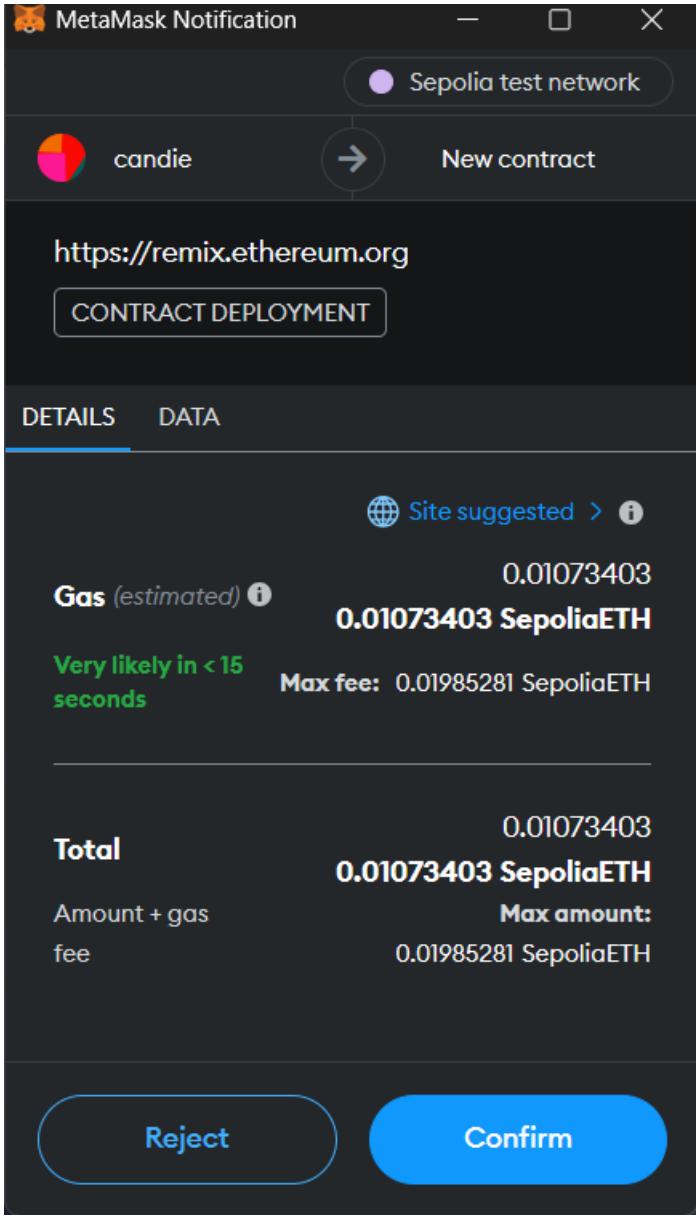
User Experience: To promote participation and avoid voter annoyance, it is essential to provide a seamless and user-friendly experience.

**Testing:**

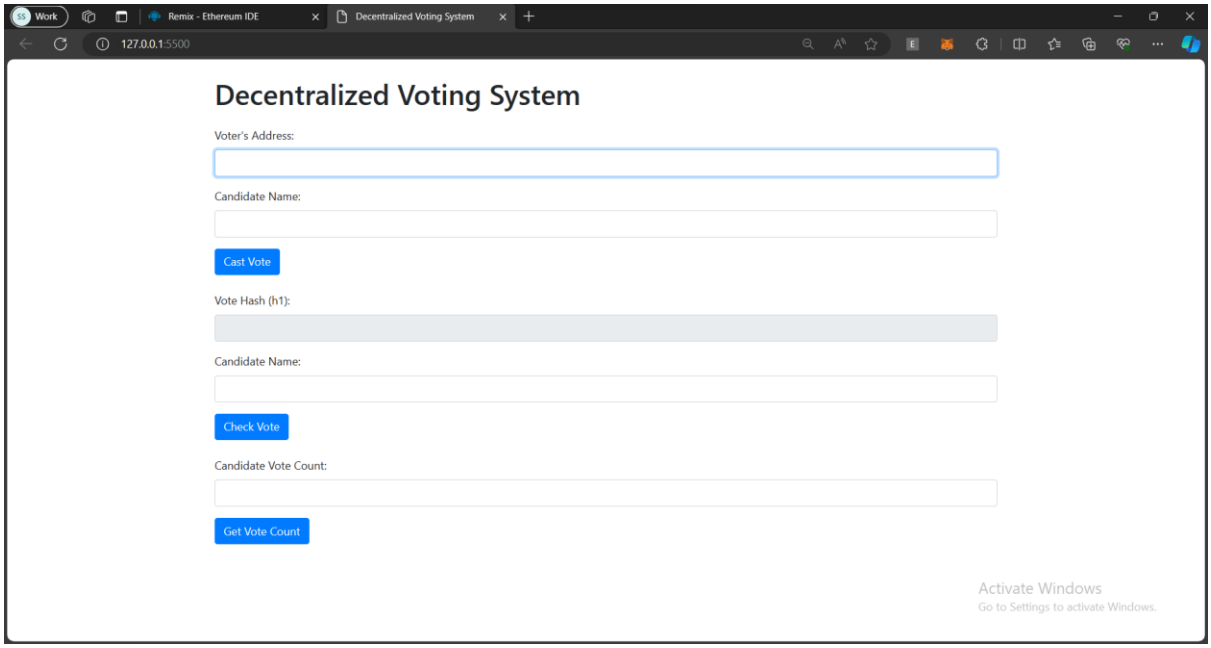
Testing procedures focused on validating the Decentralized Voting System's functionality, reliability, and security. Rigorous unit tests, integration tests, and simulated user interactions were conducted to verify the smart contracts' accuracy, frontend usability, and backend processes. Security measures included extensive auditing to identify and rectify vulnerabilities, employing encryption techniques, and ensuring secure data transmission. Testing environments replicated real-world scenarios to detect and address potential issues. Continuous monitoring and code review contributed to the system's robustness against threats, ensuring a secure and trustworthy voting platform for participants.

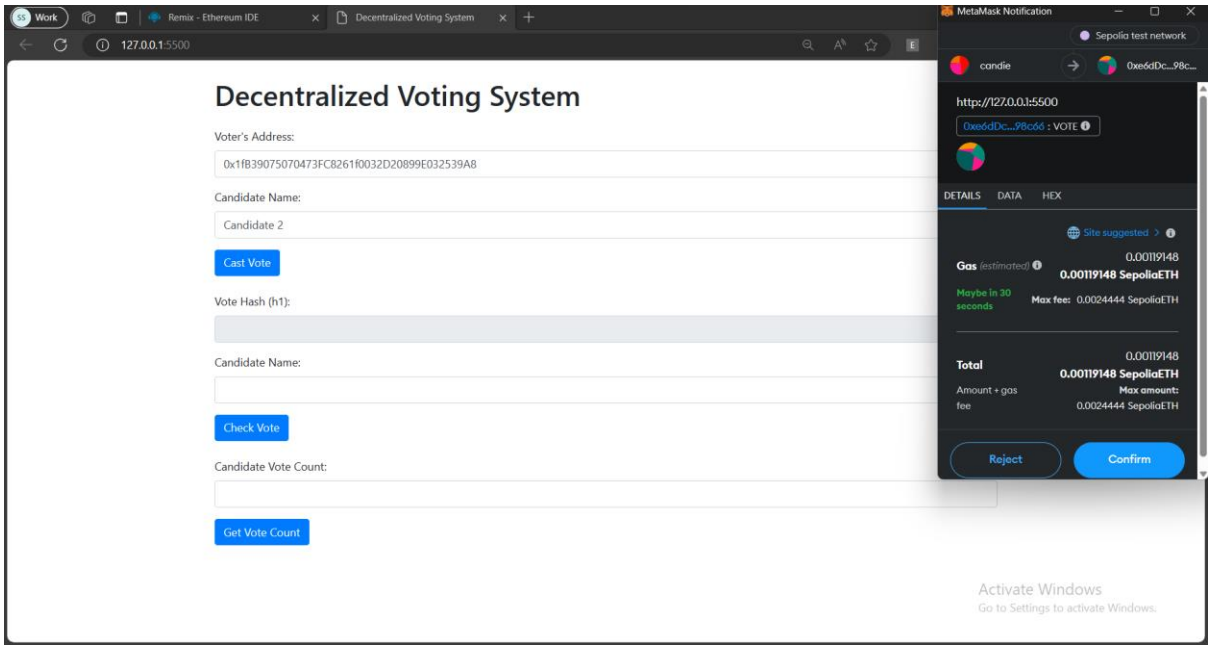
# Results and Findings:

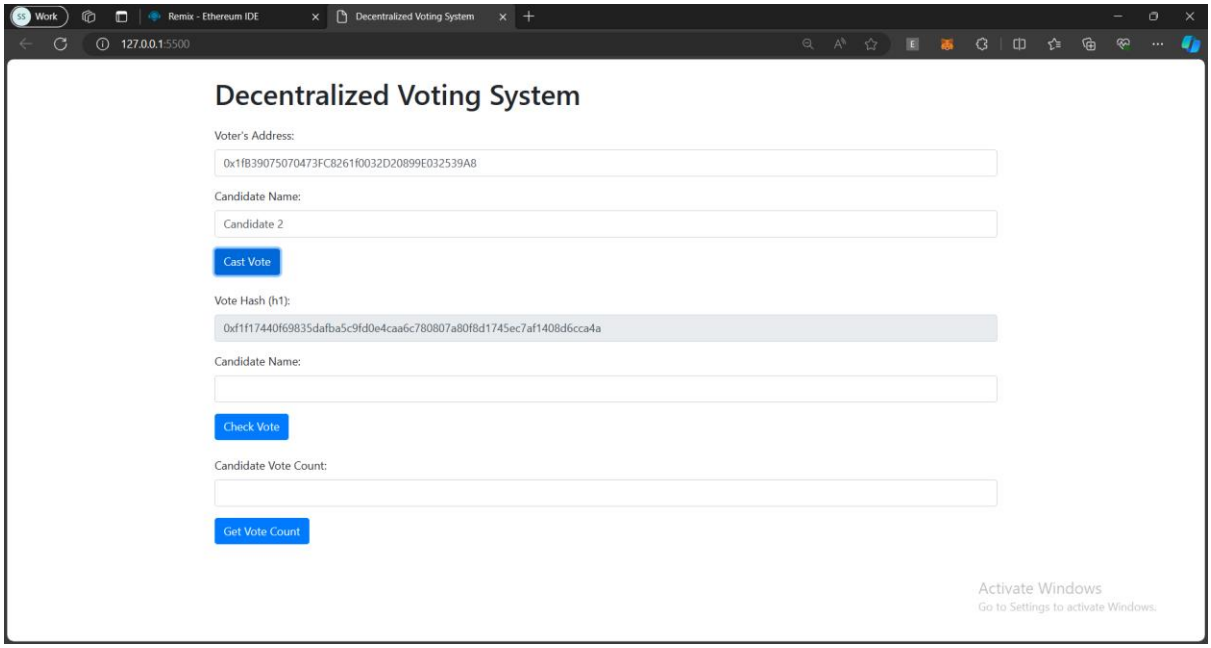


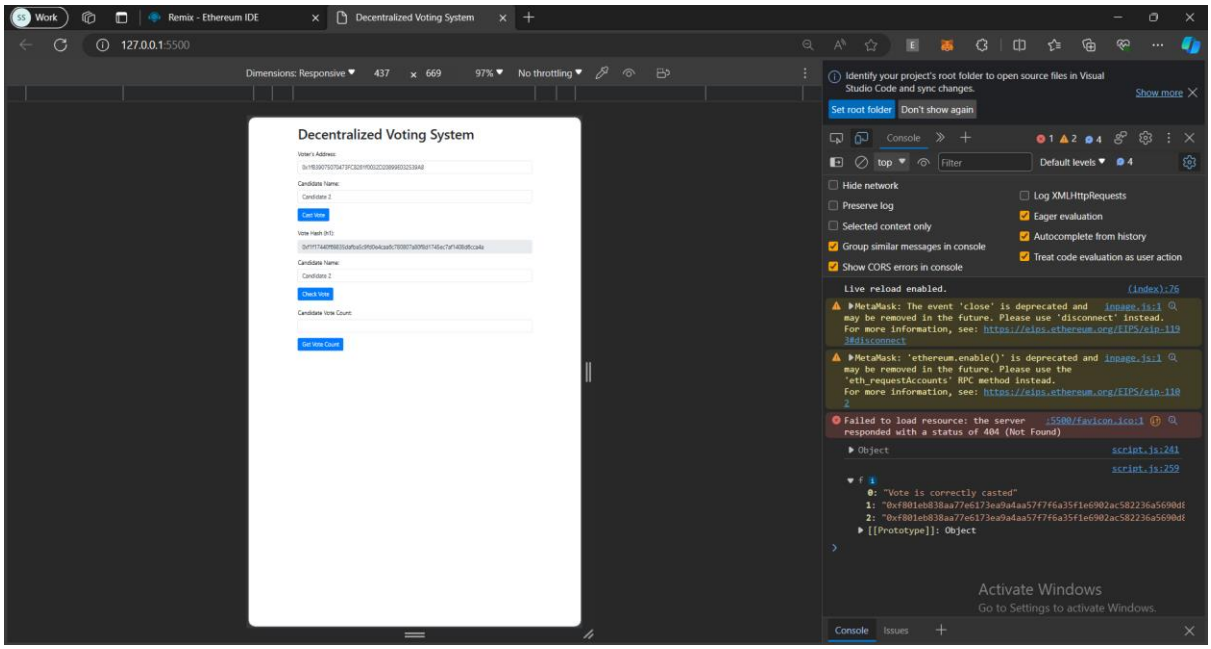

MetaMask Notification
Sepolia test network
candie
New contract
https://remix.ethereum.org
CONTRACT DEPLOYMENT
DETAILS
DATA
Site suggested
Gas (estimated)
0.01073403
0.01073403 SepoliaETH
Very likely in < 15 seconds
Max fee: 0.01985281 SepoliaETH
Total
0.01073403
0.01073403 SepoliaETH
Amount + gas fee
Max amount:
0.01985281 SepoliaETH
Reject
Confirm

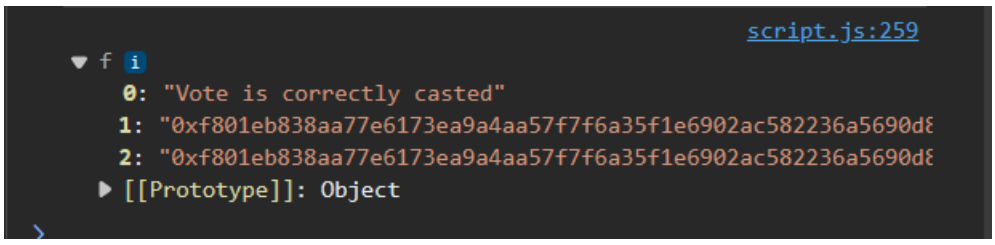

view on etherscan
[block:4738648 txIndex:22] from: 0x1fb...539a8 to: Voting.(constructor) value: 0 wei data: 0x608...60033 logs: 0 hash: 0x451...70b41


Work
Remix - Ethereum IDE
Decentralized Voting System
127.0.0.1:5500
Decentralized Voting System
Voter's Address:
Candidate Name:
Cast Vote
Vote Hash (h1):
Candidate Name:
Check Vote
Candidate Vote Count:
Get Vote Count
Activate Windows
Go to Settings to activate Windows.


Work
Remix - Ethereum IDE
Decentralized Voting System
127.0.0.1:5500
Decentralized Voting System
Voter's Address:
0x1fB39075070473FC8261f0032D20899E032539A8
Candidate Name:
Candidate 2
Cast Vote
Vote Hash (h1):
Candidate Name:
Check Vote
Candidate Vote Count:
Get Vote Count
Activate Windows
Go to Settings to activate Windows.
MetaMask Notification
Sepolia test network
candie
0xe6dDc...98c...
http://127.0.0.1:5500
0xe6dDc...98c66 : VOTE
DETAILS
DATA
HEX
Site suggested
Gas (estimated)
0.00119148
0.00119148 SepoliaETH
Maybe in 30 seconds
Max fee: 0.0024444 SepoliaETH
Total
0.00119148
0.00119148 SepoliaETH
Amount + gas fee
Max amount:
0.0024444 SepoliaETH
Reject
Confirm


Work
Remix - Ethereum IDE
Decentralized Voting System
127.0.0.1:5500
Decentralized Voting System
Voter's Address:
0x1fB39075070473FC8261f0032D20899E032539A8
Candidate Name:
Candidate 2
Cast Vote
Vote Hash (h1):
0xf1f17440f69835dafba5c9fd0e4caa6c780807a80f8d1745ec7af1408d6cca4a
Candidate Name:
Check Vote
Candidate Vote Count:
Get Vote Count
Activate Windows
Go to Settings to activate Windows.


Work
Remix - Ethereum IDE
Decentralized Voting System
127.0.0.1:5500
Dimensions: Responsive 437 x 669 97% No throttling
Identify your project's root folder to open source files in Visual Studio Code and sync changes.
Show more
Set root folder
Don't show again
Console
Hide network
Preserve log
Selected context only
Group similar messages in console
Show CORS errors in console
Log XMLHttpRequests
Eager evaluation
Autocomplete from history
Treat code evaluation as user action
Live reload enabled.
(index):76
MetaMask: The event 'close' is deprecated and may be removed in the future. Please use 'disconnect' instead.
MetaMask: 'ethereum.enable()' is deprecated and may be removed in the future. Please use the 'eth_requestAccounts' RPC method instead.
Failed to load resource: the server responded with a status of 404 (Not Found)
script.js:241
script.js:259
0: "Vote is correctly casted"
[[Prototype]]: Object
Activate Windows
Go to Settings to activate Windows.
Console
Issues


script.js:259
f
0: "Vote is correctly casted"
1: "0xf801eb838aa77e6173ea9a4aa57f7f6a35f1e6902ac582236a5690d
2: "0xf801eb838aa77e6173ea9a4aa57f7f6a35f1e6902ac582236a5690d
[[Prototype]]: Object

**Conclusion:**

In conclusion, the development and implementation of the Decentralized Voting System using Ethereum's blockchain, Solidity, and Web3 technology signify a significant stride towards transparent and secure voting mechanisms. The project successfully demonstrated the potential of blockchain in reshaping democratic processes, mitigating common issues encountered in centralized voting systems. The system's integrity, facilitated by smart contracts, ensured tamper-resistant and verifiable voting transactions. Web3 technology enabled user-friendly interactions, enhancing accessibility. Challenges, including scalability considerations and security optimizations, were addressed through meticulous testing and iterative refinements. This project contributes to the discourse on leveraging decentralized technologies for enhancing democratic practices. While achieving its objectives, this system serves as a foundation for further advancements, encouraging the adoption of decentralized voting systems in real-world scenarios, fostering trust, transparency, and inclusivity in electoral processes.

**Reference and understandings:**


1) **https://doi.org/10.1109/ACCESS.2019.2936094**
   Understanding of blockchain implementation and perspective of deploying DAPP application.

2) [2203.09738] Blockchain for the Metaverse: A Review (arxiv.org)
   Understanding of metaverse “a virtual, interconnected, and immersive digital universe or space where people can interact with each other, digital objects, and environments in real-time. It's a concept often associated with science fiction, but in recent years, it has gained significant attention and development in the tech and gaming industries.” , Digital Ownership and Scarcity, Digital identity.
3) Blockchain for decentralization of internet: prospects, trends, and challenges | Cluster Computing (springer.com)
   Understanding of decentralization and challenges, independent ledger technology.
4) [2012.10253] Data Storage in the Decentralized World: Blockchain and Derivatives (arxiv.org)
   Data Storage in decentralized network.
5) Ethereum Yellow Paper: a formal specification of Ethereum, a programmable blockchain (cryptodeep.ru)
   Understanding of ethereum, smart contract deployment and working of smart contract in decentralized network.
6) Towards analyzing the complexity landscape of solidity based ethereum smart contracts | Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (acm.org)
   Analyzing the working of solidity and working of solidity smart contract in ethereum network.
7) An overview of Ethereum and Solidity vulnerabilities | IEEE Conference Publication | IEEE Xplore
   Understanding of complexity involved while using smart contract through solidity programming language.

8) Smart contracts: security patterns in the ethereum ecosystem and solidity | IEEE Conference Publication | IEEE Xplore
Fining patterns of ethereum ecosystem and solidity
9) Ethereum Smart Contract Development: Build blockchain-based decentralized ... - Mayukh Mukhopadhyay - Google Books
Structured programming way of implementing decentralized application using solidity.
10) Decentralized Voting: A Self-tallying Voting System Using a Smart Contract on the Ethereum Blockchain | SpringerLink
Experiments of implementing voting system smart contract on the ethereum block chain.
11) ORBilu: Detailled Reference (uni.lu)
Fair development of robust and structed voting software through the network.
12) 68819.pdf (scitepress.org)
Understanding of vulnerability of implementing decentralized voting machine in blockchain.
13) A Comparitive Analysis on E-Voting System Using Blockchain | IEEE Conference Publication | IEEE Xplore
Compartive analysis of future implementation of E-voting system by the block chain network.