

SCHOOL OF INFORMATION, COMPUTER AND COMMUNICATION
TECHNOLOGY

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THAMMASAT UNIVERSITY

CSS486 Blockchain Development

**Group Project: Secure Voting System Smart Contract and
Decentralized Application**

By

Patsita thaweeratsirichot	6222771576
Jiranut Sukkee	6222781823
Pakpum Yu	6222781849

Advisor: Dr. Watthanasak Jeamwatthanachai

Table of Content

Chapter 1: Introduction	3
Chapter 2: Design Specification	4
Chapter 3: Methodology	7
Chapter 4: Functionality Test and Verification	15
Chapter 5: Conclusion	19
References	

Chapter 1: Introduction

Smart contracts and distributed ledger technology (DLT) can be used to create a trustworthy and auditable voting system. By using smart contracts on a blockchain, we can create trustworthy voting systems where only verified voters can cast ballots and trustworthy outcomes can be reliably tabulated. They can also guarantee the integrity of the vote by making voter preferences unchangeable. Traditional voting systems are plagued by issues such as voter fraud, manipulation, and a lack of confidence in the tabulation process. Blockchain technology and the development of decentralized applications can aid in overcoming these issues. In this update, we will investigate the viability of utilizing smart contracts and decentralized applications to build a more secure and transparent voting system.

Chapter 2: Design Specification

2.1 System Architecture

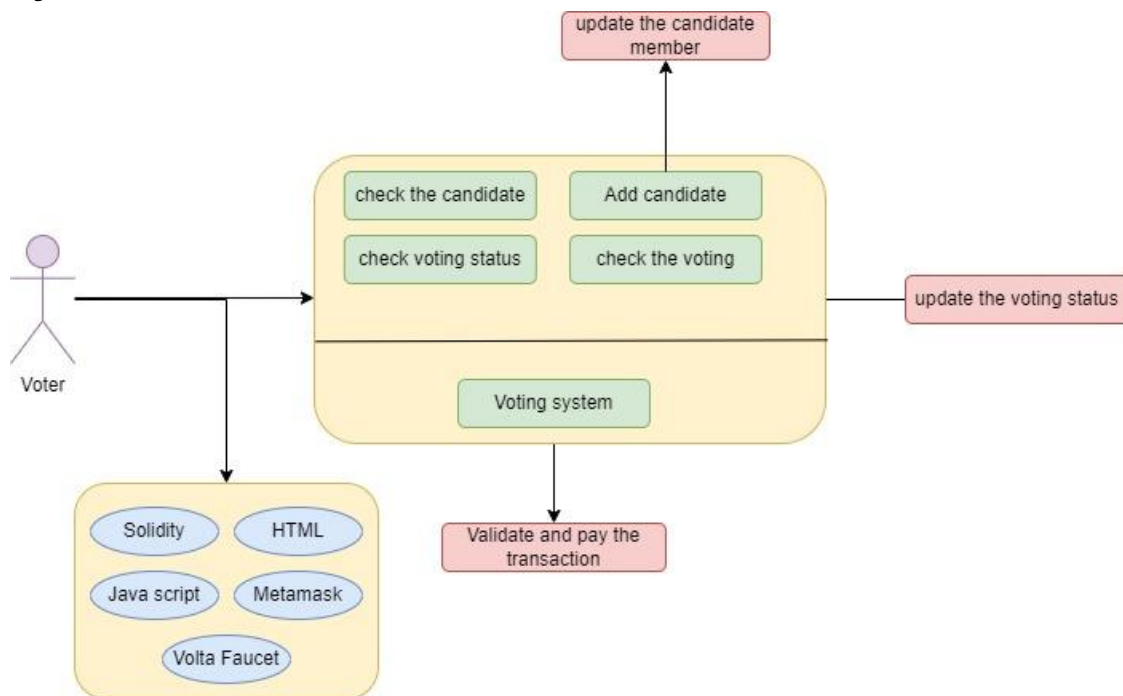


Figure 2.1

2.1.1 Browser

Any browser can access our website. Such as firefox, google chrome, safari, Microsoft edge that can add Metamask etc.

2.1.2 Secure Voting System Smart Contract

When a user enters the website, the user should be able to connect the metamask before voting and adding the candidate. Before the voting was updated the candidate user must pay the transaction on the metamask. After we vote for the candidate we can check all the numbers of voting for each candidate.

2.1.3 Payment Gateway

Payment for the voter is just the only transaction fee from voting by paying on Metamask.

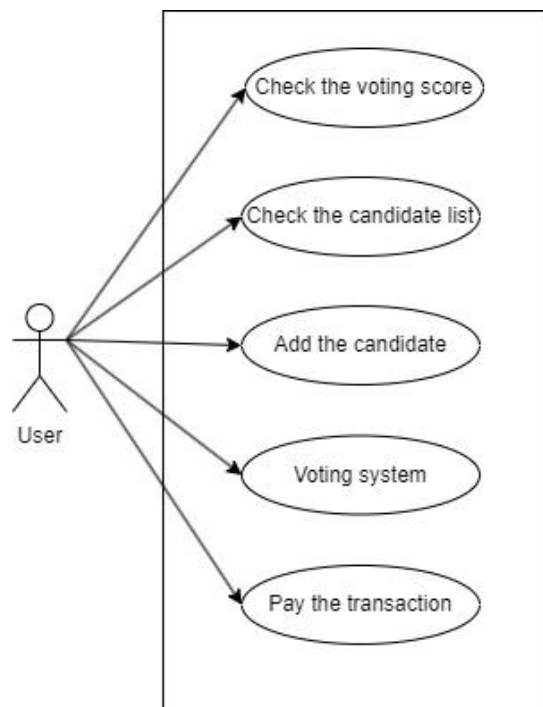
2.1.4 Tools

This system uses Solidity to create Smart contracts, JavaScript and HTML for websites and Volta is for testnet faucet to pay transactions.

2.2 Detail design

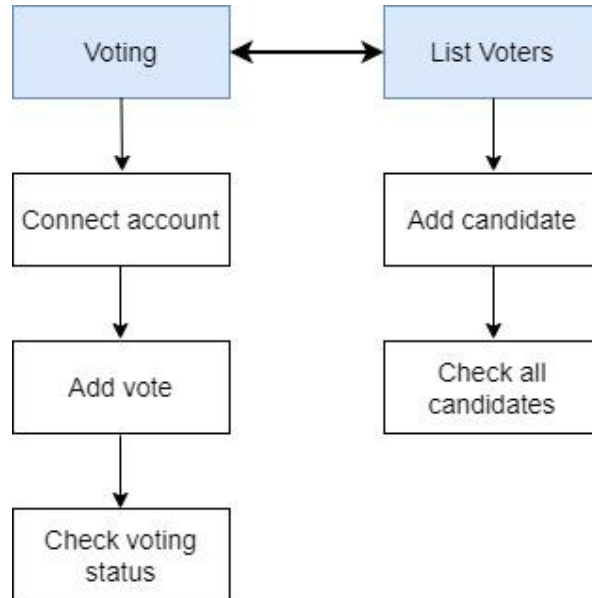
2.2.1 Use case Diagram

This following diagram shows the usage case of all users in this website.



2.2.2 Prototype

Sitemap



Chapter 3: Methodology

For dApp secure Voting system was written to test the system of voting in smart contract, before we use and for the function we focus on the main system of voting are add candidate, voting, check the score and make it stable. For our blockchain we use Ethereum smart contract and our main function is Smart contract that uses solidity. For front-end coding is index.js and ListVoter.js and main.js is about functions that make Voting on the dApp works on the website. The last is deploy.js is a code that work for connect to the metamask.

This is all our main source code in the voting smart contract:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

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

    Candidate[] public candidates;
    address owner;
    mapping(address => bool) public voters;

    uint256 public votingStart;
    uint256 public votingEnd;

    constructor(string[] memory _candidateNames, uint256 _durationInMinutes) {
        for (uint256 i = 0; i < _candidateNames.length; i++) {
            candidates.push(Candidate({
                name: _candidateNames[i],
                voteCount: 0
            }));
        }
        owner = msg.sender;
        votingStart = block.timestamp;
        votingEnd = block.timestamp + (_durationInMinutes * 1 minutes);
    }

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

Figure 3.1.1 Voting.sol

```

function addCandidate(string memory _name) public onlyOwner {
    candidates.push(Candidate({
        name: _name,
        voteCount: 0
    }));
}

function vote(uint256 _candidateIndex) public {
    require(!voters[msg.sender], "You have already voted.");
    require(_candidateIndex < candidates.length, "Invalid candidate index.");

    candidates[_candidateIndex].voteCount++;
    voters[msg.sender] = true;
}

function getAllVotesOfCandidates() public view returns (Candidate[] memory){
    return candidates;
}

function getVotingStatus() public view returns (bool) {
    return (block.timestamp >= votingStart && block.timestamp < votingEnd);
}

function getRemainingTime() public view returns (uint256) {
    require(block.timestamp >= votingStart, "Voting has not started yet.");
    if (block.timestamp >= votingEnd) {
        return 0;
    }
    return votingEnd - block.timestamp;
}

```

Figure 3.1.2 Voting. sol


```

</head>
<body>
  <div class = "navbar">
    <a href="/index.html">Vote</a>
    <a href="/ListVoters.html">List Voters</a>
  </div>

  <div class="container1" style="background-image: url('bg.jpeg');">
    Welcome to the Decentralized Voting Application</div>

  <div>
    <button onclick="connectMetamask()">Connect Account</button>
    <p id="metamasknotification"></p>
  </div>

  <div class="container">
    <form method="POST" action="/vote" enctype="multipart/form-data">
      <span>Add candidate here</span>
      <input type = "text" name ="vote" placeholder="Type name of candidate ... ">
      <input type="submit" value = "Add">
    </form>
  </div>

  <div>
    <p id= "votingStatus"></p>
  </div>

  <div>
    <button onclick="getAllCandidates()">Check All Candidates</button>
    <p id="p3"></p>
  </div>

```

Figure 3.2.1 ListVoter.html

```
<div>
  <table id="myTable">
    <thead>
      <tr>
        <th>Index</th>
        <th>Candidate name</th>
        <th>Vote Number</th>
      </tr>
    </thead>
    <tbody>

    </tbody>
  </table>
</div>

</body>
</html>
```

Figure 3.2.2 ListVoter.html

```

</head>
<body>
  <div class = "navbar">
    <a href="/index.html">Vote</a>
    <a href="/ListVoters.html">List Voters</a>
  </div>

  <div class="container1" style="background-image: url('bg.jpeg');">
    Welcome to the Decentralized Voting Application</div>
  <div class="row">
    <div class="col-lg-6">
      <div>
        <button onclick="connectMetamask()">Connect Account</button>
        <p id="metamasknotification"></p>
      </div>
    </div>

    <div class="col-lg-6"></div>
    <div class="container">
      <span>Vote here</span>
      <input type = "number" id ="vote" placeholder="Type Index ">
      <button onclick="addVote()">Add Vote</button>
      <p id="cand"></p>

      <button onclick="voteStatus()">Check Status of Voting</button>
      <p id = "status"></p>
      <p id = "time"></p>
    </div>
  </div>
</body>
</html>

```

Figure 3.3.1 Index.html

```

const connectMetamask = async() => {
  const provider = new ethers.providers.Web3Provider(window.ethereum);
  await provider.send("eth_requestAccounts", []);
  const signer = provider.getSigner();
  WALLET_CONNECTED = await signer.getAddress();
  var element = document.getElementById("metamasknotification");
  element.innerHTML = "Metamask is connected " + WALLET_CONNECTED;
}

const addVote = async() => {
  if(WALLET_CONNECTED !== 0) {
    var name = document.getElementById("vote");
    const provider = new ethers.providers.Web3Provider(window.ethereum);
    await provider.send("eth_requestAccounts", []);
    const signer = provider.getSigner();
    const contractInstance = new ethers.Contract(contractAddress, contractAbi, signer);
    var cand = document.getElementById("cand");
    cand.innerHTML = "Please wait, adding a vote in the smart contract";
    const tx = await contractInstance.vote(name.value);
    await tx.wait();
    cand.innerHTML = "Vote added !!!";
  }
  else {
    var cand = document.getElementById("cand");
    cand.innerHTML = "Please connect metamask first";
  }
}

```

Figure 3.4.1 main.js

```

const voteStatus = async() => {
  if(WALLET_CONNECTED !== 0) {
    var status = document.getElementById("status");
    var remainingTime = document.getElementById("time");
    const provider = new ethers.providers.Web3Provider(window.ethereum);
    await provider.send("eth_requestAccounts", []);
    const signer = provider.getSigner();
    const contractInstance = new ethers.Contract(contractAddress, contractAbi, signer);
    const currentStatus = await contractInstance.getVotingStatus();
    const time = await contractInstance.getRemainingTime();
    console.log(time);
    status.innerHTML = currentStatus == 1 ? "Voting is currently open" : "Voting is finished";
    remainingTime.innerHTML = `Remaining time is ${parseInt(time, 16)} seconds`;
  }
  else {
    var status = document.getElementById("status");
    status.innerHTML = "Please connect metamask first";
  }
}

```

Figure 3.4.2 main.js

```

const getAllCandidates = async() => {
  if(WALLET_CONNECTED !== 0) {
    var p3 = document.getElementById("p3");
    const provider = new ethers.providers.Web3Provider(window.ethereum);
    await provider.send("eth_requestAccounts", []);
    const signer = provider.getSigner();
    const contractInstance = new ethers.Contract(contractAddress, contractAbi, signer);
    p3.innerHTML = "Please wait, getting all the candidates from the voting smart contract";
    var candidates = await contractInstance.getAllVotesOfCandidates();
    console.log(candidates);
    var table = document.getElementById("myTable");

    for (let i = 0; i < candidates.length; i++) {
      var row = table.insertRow();
      var idCell = row.insertCell();
      var descCell = row.insertCell();
      var statusCell = row.insertCell();

      idCell.innerHTML = i;
      descCell.innerHTML = candidates[i].name;
      statusCell.innerHTML = candidates[i].voteCount;
    }

    p3.innerHTML = "The tasks are updated"
  }
  else {
    var p3 = document.getElementById("p3");
    p3.innerHTML = "Please connect metamask first";
  }
}

```

Figure 3.4.3 main.js

```

const hre = require("hardhat");

async function main() {
  const Voting = await hre.ethers.getContractFactory("voting");
  const Voting_ = await Voting.deploy([], 10);

  await Voting_.deployed();

  console.log(
    `Contract address : ${Voting_.address}`
  );
}

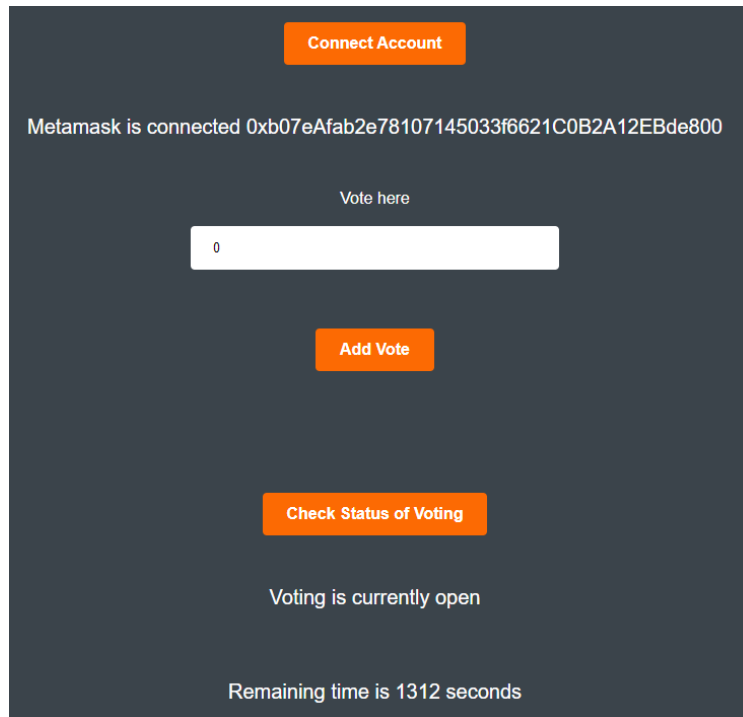
// We recommend this pattern to be able to use async/await everywhere
// and properly handle errors.
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
}));

```

Figure 3.5.1 deploy.html

Chapter 4: Functionality Test and Verification

This is our function in dApp voting smart contract, how it works and verification.



The screenshot displays a dark-themed user interface for a dApp voting system. At the top, an orange button labeled "Connect Account" is visible. Below it, a message states "Metamask is connected 0xb07eAfab2e78107145033f6621C0B2A12EBde800". A "Vote here" label is positioned above a white input field containing the number "0". Below the input field is an orange "Add Vote" button. Further down is an orange "Check Status of Voting" button. The status section shows "Voting is currently open" and "Remaining time is 1312 seconds".

Figure 4.1 Connect the metamask and Check the time remaining while in the voting.

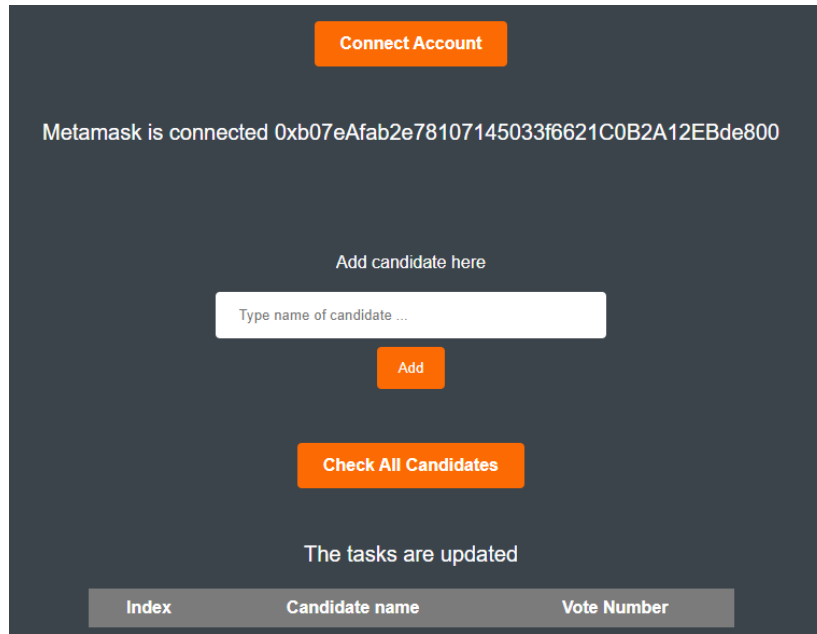


Figure 4.2 We come to ListVoter and connect account again and check all candidates before we votes.

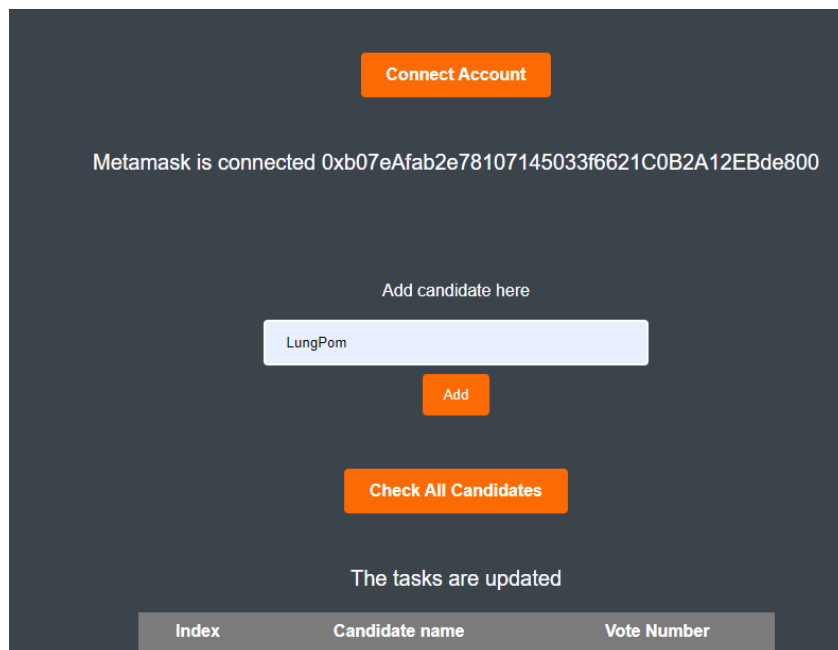


Figure 4.3 Then if it doesn't have any candidates. Add the candidates and press the add button.

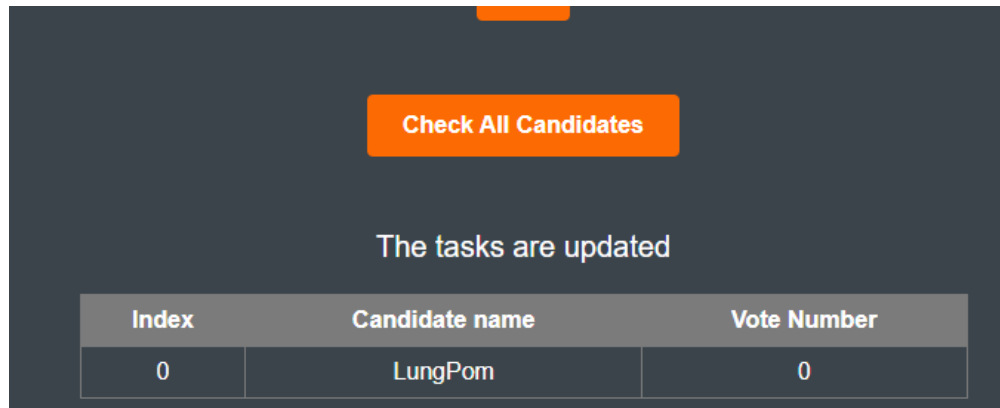


Figure 4.4 We press the checked all candidates button to check the number of candidates name and we move to the votes pages.

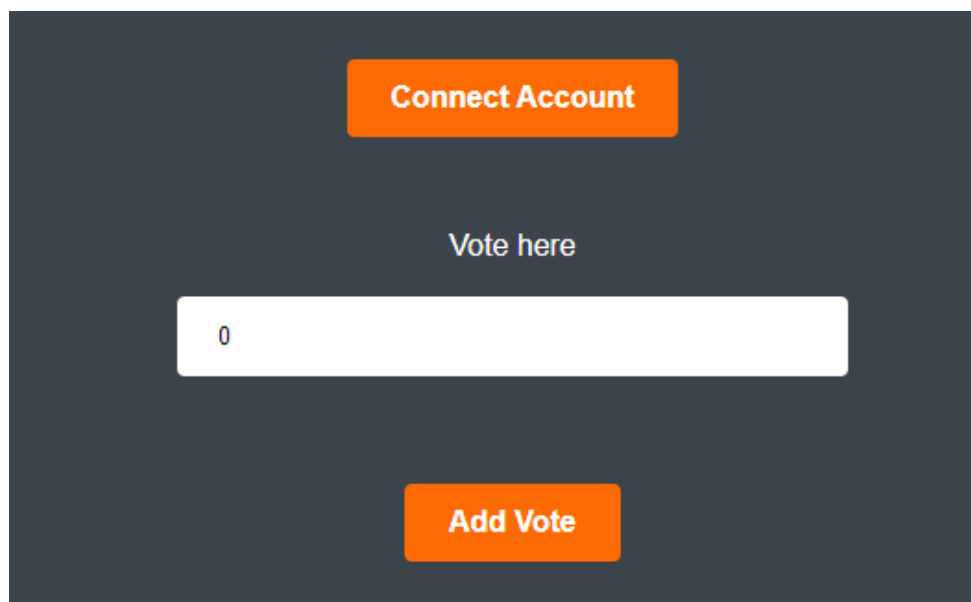


Figure 4.5 We choose the vote number from the table of candidates and then click Add Vote.

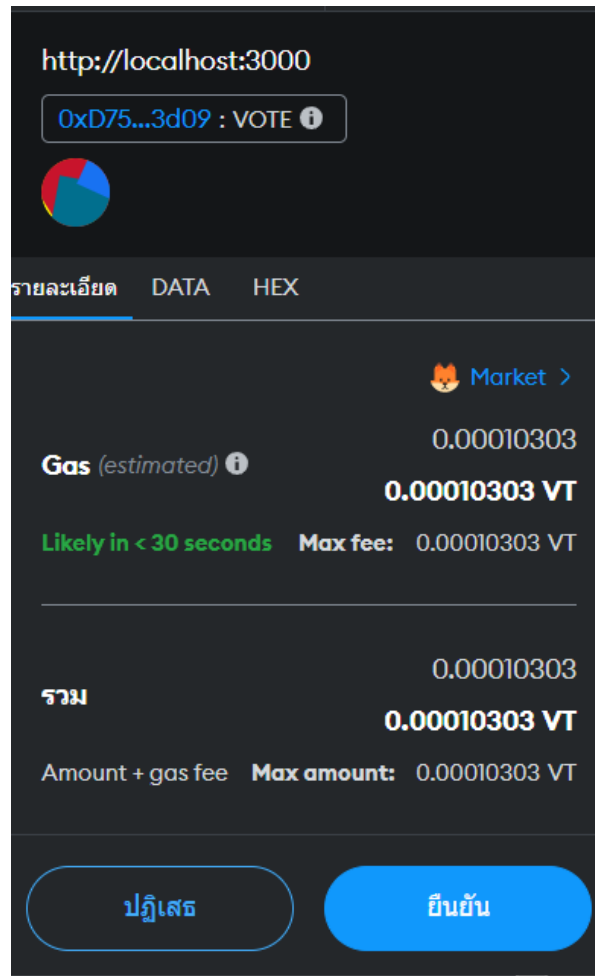


Figure 4.6 Pay the transaction before you enter the vote.

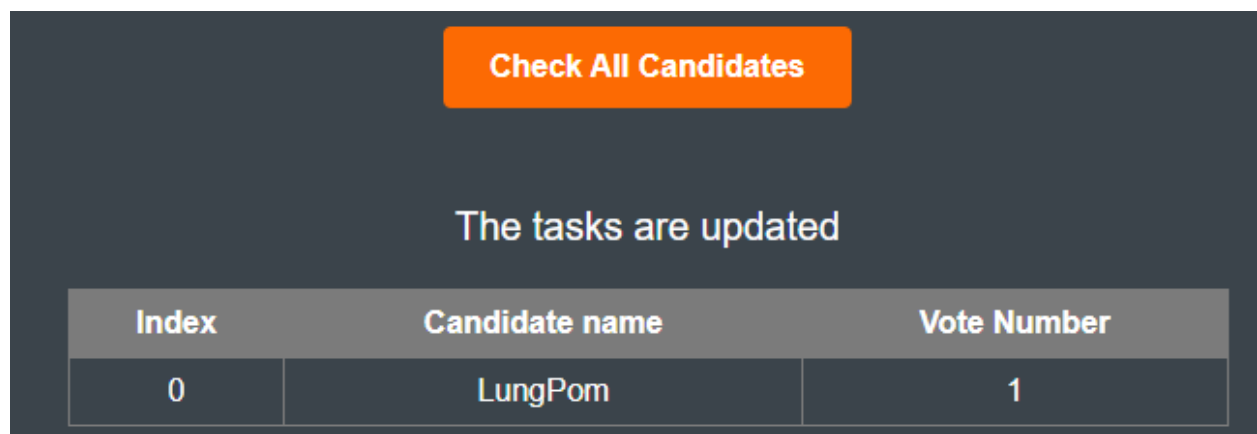


Figure 4.7 Then we come to page ListVoters, connect Metamask and click Check All Candidates and wait for the result.

Chapter 5: Conclusion

In conclusion, smart contracts have mitigated fraud by using blockchain technology. Deterministic code execution and transparent transaction recording have also solved the verification problem. Smart contracts also improve trust and equity by increasing transparency. Contractual processes are now more efficient and reliable, enabling a more secure and transparent digital economy.

References

[1] "Jan 5, 2022 · I have tried playing with hardhat gas & gasprice setting but with no succes. The code async function main() { const MyNFT = await ethers.getContractFactory("MyPRODNFT") const myNFT = await MyNFT.deploy() console.log("Contract deployed to address:", myNFT.address) } main() .then(() => process.exit(0)) .catch((error) => { console.error ..."

URL:

<https://ethereum.stackexchange.com/questions/118086/constantly-getting-insufficient-funds-for-intrinsic-transaction-cost>

[3] "Feb 27, 2021 · Solidity ParserError: Expected ';' but got '{'. pragma solidity ^0.6.0; contract Test { function sendValue (address payable recipient, uint256 amount) external { (bool success,) = recipient.call { value: amount } (""); } }

URL:

<https://stackoverflow.com/questions/66388642/solidity-parsererror-expected-but-got>