**Project 5: Voting with Ethereum Blockchain**

**By**

**REVATHI S**

**REQUIREMENTS :**

**Smart Contract:**

* Solidity smart contract named "Voting".
* Constructor with candidate names and duration.
* Owner can add candidates using addCandidate function.
* vote function for users to cast votes.
* Track votes for each candidate in candidates array.
* Determine voting status (open or finished).
* Function to calculate remaining voting time.
* Retrieve list of candidates and their votes.

**Frontend (HTML and JavaScript):**

* HTML interfaces for interaction.
* Connect Metamask wallet.
* Allow users to submit votes.
* Display voting status prominently.
* Show remaining voting time.
* List candidates with vote counts.

**JavaScript (main.js):**

* Use ethers.js for Ethereum interaction.
* connectMetamask to connect wallet.
* addVote for user voting.
* voteStatus to display status and time.
* getAllCandidates to list candidates.

**Deployment Configuration (index.js and .env):**

* Express.js server for handling requests.
* Routes for serving files and actions.
* Use environment variables for sensitive data.
* Configure server to listen on port.

**OVERVIEW OF THE CODE :**

1. **HTML Frontend (index.html and ListVoters.html):**
   * These HTML files provide the user interface for your voting application.
   * Users can connect their Metamask wallet, add votes for candidates, check the voting status, and view a list of candidates and their vote counts.
   * Elements like buttons, input fields, and text areas are used to interact with the Ethereum blockchain.
2. **JavaScript Logic (main.js):**
   * This JavaScript file contains various functions that interact with the Ethereum blockchain and update the user interface.
   * connectMetamask(): Connects Metamask wallet and displays the connected address.
   * addVote(): Allows users to add their votes for a specific candidate.
   * voteStatus(): Displays the current voting status (open or finished) and remaining time.
   * getAllCandidates(): Retrieves all candidates and their vote counts from the smart contract.
3. **Smart Contract (Voting.sol):**
   * This Solidity contract defines the behavior of the voting system on the Ethereum blockchain.
   * It has functions to add candidates, vote for candidates, retrieve voting status, remaining time, and all candidate information.
   * The contract is deployed to the Ethereum network and interacts with the JavaScript frontend through its ABI and contract address.
4. **Deployment Configuration (.env and index.js):**
   * The .env file contains environment variables like RPC\_URL, PRIVATE\_KEY, CONTRACT\_ADDRESS, etc.
   * The index.js file uses these environment variables to set up the Express server and interact with the smart contract using ethers.js.
   * The server listens on a specific port and handles HTTP requests, such as adding votes and retrieving candidate information.
5. **Miscellaneous:**
   * The code handles connecting to Metamask, interacting with the smart contract, and displaying relevant information on the frontend.
   * It uses ethers.js to create a Web3Provider and interact with the Ethereum blockchain.

**SOURCE CODE:**

**VOTING.SOL**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

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);

\_;

}

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 getAllVotesOfCandiates() 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;

}

}

**MAIN.JS:**

let WALLET\_CONNECTED = "";

let contractAddress = "0xXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";

let contractAbi = [

  {

    inputs: [

      {

        internalType: "string[]",

        name: "\_candidateNames",

        type: "string[]",

      },

      {

        internalType: "uint256",

        name: "\_durationInMinutes",

        type: "uint256",

      },

    ],

    stateMutability: "nonpayable",

    type: "constructor",

  },

  {

    inputs: [

      {

        internalType: "string",

        name: "\_name",

        type: "string",

      },

    ],

    name: "addCandidate",

    outputs: [],

    stateMutability: "nonpayable",

    type: "function",

  },

  {

    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: [],

    name: "getAllVotesOfCandiates",

    outputs: [

      {

        components: [

          {

            internalType: "string",

            name: "name",

            type: "string",

          },

          {

            internalType: "uint256",

            name: "voteCount",

            type: "uint256",

          },

        ],

        internalType: "struct Voting.Candidate[]",

        name: "",

        type: "tuple[]",

      },

    ],

    stateMutability: "view",

    type: "function",

  },

  {

    inputs: [],

    name: "getRemainingTime",

    outputs: [

      {

        internalType: "uint256",

        name: "",

        type: "uint256",

      },

    ],

    stateMutability: "view",

    type: "function",

  },

  {

    inputs: [

      {

        internalType: "uint256",

        name: "\_candidateIndex",

        type: "uint256",

      },

    ],

    name: "getVotesOfCandiate",

    outputs: [

      {

        internalType: "uint256",

        name: "",

        type: "uint256",

      },

    ],

    stateMutability: "view",

    type: "function",

  },

  {

    inputs: [],

    name: "getVotingStatus",

    outputs: [

      {

        internalType: "bool",

        name: "",

        type: "bool",

      },

    ],

    stateMutability: "view",

    type: "function",

  },

  {

    inputs: [

      {

        internalType: "uint256",

        name: "\_candidateIndex",

        type: "uint256",

      },

    ],

    name: "vote",

    outputs: [],

    stateMutability: "nonpayable",

    type: "function",

  },

  {

    inputs: [

      {

        internalType: "address",

        name: "",

        type: "address",

      },

    ],

    name: "voters",

    outputs: [

      {

        internalType: "bool",

        name: "",

        type: "bool",

      },

    ],

    stateMutability: "view",

    type: "function",

  },

  {

    inputs: [],

    name: "votingEnd",

    outputs: [

      {

        internalType: "uint256",

        name: "",

        type: "uint256",

      },

    ],

    stateMutability: "view",

    type: "function",

  },

  {

    inputs: [],

    name: "votingStart",

    outputs: [

      {

        internalType: "uint256",

        name: "",

        type: "uint256",

      },

    ],

    stateMutability: "view",

    type: "function",

  },

];

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";

  }

};

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";

  }

};

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.getAllVotesOfCandiates();

    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";

  }

};

**INDEX.JS**

require("dotenv").config();

const express = require("express");

const app = express();

const fileUpload = require("express-fileupload");

app.use(

  fileUpload({

    extended: true,

  })

);

app.use(express.static(\_\_dirname));

app.use(express.json());

const path = require("path");

const ethers = require("ethers");

var port = 3000;

const RPC\_URL = process.env.RPC\_URL;

const PRIVATE\_KEY = process.env.PRIVATE\_KEY;

const CONTRACT\_ADDRESS = process.env.CONTRACT\_ADDRESS;

const { abi } = require("./artifacts/contracts/Voting.sol/Voting.json");

const provider = new ethers.providers.JsonRpcProvider(RPC\_URL);

const signer = new ethers.Wallet(PRIVATE\_KEY, provider);

const contractInstance = new ethers.Contract(CONTRACT\_ADDRESS, abi, signer);

app.get("/", (req, res) => {

  res.sendFile(path.join(\_\_dirname, "index.html"));

});

app.get("/index.html", (req, res) => {

  res.sendFile(path.join(\_\_dirname, "index.html"));

});

app.post("/vote", async (req, res) => {

  var vote = req.body.vote;

  console.log(vote);

  async function storeDataInBlockchain(vote) {

    console.log("Adding the candidate in the voting contract...");

    const tx = await contractInstance.addCandidate(vote);

    await tx.wait();

  }

  const bool = await contractInstance.getVotingStatus();

  if (bool == true) {

    await storeDataInBlockchain(vote);

    res.send("The candidate has been registered in the smart contract");

  } else {

    res.send("Voting is finished");

  }

});

app.listen(port, function () {

  console.log("App is listening on port 3000");

});

**INDEX.HTML**

<!DOCTYPE html>

<html lang="en">

<head>

    <title>To Do Decentralized Application</title>

    <script src="main.js"></script>

    <script src="https://cdn.ethers.io/lib/ethers-5.7.1.umd.min.js"

        type="application/javascript"></script>

    <title>Voting DAPP</title>

    <style>

        button {

            background-color: #4CAF50;

            color: white;

            padding: 12px 24px;

            border: none;

            border-radius: 4px;

            cursor: pointer;

            font-size: 16px;

            font-weight: bold;

            display: block;

            margin: 50px auto 0;

        }

        button:hover {

            background-color: #3e8e41;

        }

        /\* Paragraph styles \*/

        p {

            text-align: center;

            font-size: 20px;

            margin-top: 50px;

        }

        form {

            display: flex;

            flex-direction: column;

            align-items: center;

            justify-content: center;

            margin-top: 50px;

        }

        input[type=text], select {

            width: 30%;

            padding: 12px 20px;

            margin: 8px 0;

            box-sizing: border-box;

            border: 2px solid #ccc;

            border-radius: 4px;

        }

        input[type=number], select {

            width: 30%;

            padding: 12px 20px;

            margin: 8px 0;

            box-sizing: border-box;

            border: 2px solid #ccc;

            border-radius: 4px;

        }

        input[type=submit] {

            background-color: #4CAF50;

            color: white;

            padding: 12px 20px;

            border: none;

            border-radius: 4px;

            cursor: pointer;

        }

        input[type=submit]:hover {

            background-color: #45a049;

        }

        .container {

            display: flex;

            flex-direction: column;

            align-items: center;

            justify-content: center;

            margin-top: 50px;

        }

        span {

            display: block;

            margin-bottom: 10px;

        }

/\* Style the navigation bar \*/

        .navbar {

            overflow: hidden;

            background-color: #333;

            font-family: Arial;

        }

        /\* Style the links inside the navigation bar \*/

        .navbar a {

            float: left;

            display: block;

            color: white;

            text-align: center;

            padding: 14px 20px;

            text-decoration: none;

        }

        /\* Change the color of links on hover \*/

        .navbar a:hover {

            background-color: #ddd;

            color: black;

        }

        #myTable {

        border-collapse: collapse;

        width: 100%;

        max-width: 600px;

        margin: 0 auto;

        }

        #myTable th,

        #myTable td {

        border: 1px solid black;

        padding: 8px;

        text-align: center;

        }

        #myTable th {

        background-color: #f2f2f2;

        font-weight: bold;

        }

    </style>

</head>

<body>

    <div class = "navbar">

        <a href="/index.html">Vote</a>

        <a href="/ListVoters.html">List Voters</a>

    </div>

    <div class="container">Welcome to the Decentralized Voting Application</div>

    <div>

        <button onclick="connectMetamask()">Connect Metamask</button>

        <p id="metamasknotification"></p>

    </div>

    <div class="container">

        <span>Vote here</span>

        <input type = "number" id ="vote" placeholder="Add Index of the Voter ... ">

        <button onclick="addVote()">Add Vote</button>

        <p id="cand"></p>

        <button onclick="voteStatus()">Check Voting Status</button>

        <p id = "status"></p>

        <p id = "time"></p>

    </div>

</body>

</html>

**HARDHAT.CONFIG.JS**

require("@nomicfoundation/hardhat-toolbox");

require("dotenv").config();

/\*\* @type import('hardhat/config').HardhatUserConfig \*/

module.exports = {

  solidity: "0.8.18", // Note: The correct format is "0.8.18", not "0.8 \* 0.18"

  networks: {

    mumbai: {

      url: process.env.RPC\_URL,

      accounts: [process.env.PRIVATE\_KEY],

    },

  },

  etherscan: {

    apiKey: process.env.API\_KEY,

  },

};

**LISTVOTERS.HTML**

<!DOCTYPE html>

<html lang="en">

<head>

    <title>To Do Decentralized Application</title>

    <script src="main.js"></script>

    <script src="https://cdn.ethers.io/lib/ethers-5.7.1.umd.min.js"

        type="application/javascript"></script>

    <title>Centered Form</title>

    <style>

button {

            background-color: #4CAF50;

            color: white;

            padding: 12px 24px;

            border: none;

            border-radius: 4px;

            cursor: pointer;

            font-size: 16px;

            font-weight: bold;

            display: block;

            margin: 50px auto 0;

        }

        button:hover {

            background-color: #3e8e41;

        }

        /\* Paragraph styles \*/

        p {

            text-align: center;

            font-size: 20px;

            margin-top: 50px;

        }

        form {

            display: flex;

            flex-direction: column;

            align-items: center;

            justify-content: center;

            margin-top: 50px;

        }

        input[type=text], select {

            width: 30%;

            padding: 12px 20px;

            margin: 8px 550px;

            box-sizing: border-box;

            border: 2px solid #ccc;

            border-radius: 4px;

        }

        input[type=number], select {

            width: 100%;

            padding: 12px 20px;

            margin: 8px 0;

            box-sizing: border-box;

            border: 2px solid #ccc;

            border-radius: 4px;

        }

        input[type=submit] {

            background-color: #4CAF50;

            color: white;

            padding: 12px 20px;

            border: none;

            border-radius: 4px;

            cursor: pointer;

        }

        input[type=submit]:hover {

            background-color: #45a049;

        }

        .container {

            display: flex;

            flex-direction: column;

            align-items: center;

            justify-content: center;

            margin-top: 50px;

        }

        span {

            display: block;

            margin-bottom: 10px;

        }

/\* Style the navigation bar \*/

        .navbar {

            overflow: hidden;

            background-color: #333;

            font-family: Arial;

        }

        /\* Style the links inside the navigation bar \*/

        .navbar a {

            float: left;

            display: block;

            color: white;

            text-align: center;

            padding: 14px 20px;

            text-decoration: none;

        }

        /\* Change the color of links on hover \*/

        .navbar a:hover {

            background-color: #ddd;

            color: black;

        }

        #myTable {

        border-collapse: collapse;

        width: 100%;

        max-width: 600px;

        margin: 0 auto;

        }

        #myTable th,

        #myTable td {

        border: 1px solid black;

        padding: 8px;

        text-align: center;

        }

        #myTable th {

        background-color: #f2f2f2;

        font-weight: bold;

        }

    </style>

</head>

<body>

    <div class = "navbar">

        <a href="/index.html">Vote</a>

        <a href="/ListVoters.html">List Voters</a>

    </div>

    <div>

        <button onclick="connectMetamask()">Connect Metamask</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="Add name of candidate ... ">

            <input type="submit" value = "Add Candidate">

        </form>

    </div>

    <div>

        <p id= "votingStatus"></p>

    </div>

    <div>

        <button onclick="getAllCandidates()">List All Candidates</button>

        <p id="p3"></p>

    </div>

    <div>

        <table id="myTable">

            <thead>

                <tr>

                    <th>Index</th>

                    <th>Candidate name</th>

                    <th>Candidate votes</th>

                </tr>

            </thead>

            <tbody>

            </tbody>

        </table>

    </div>

</body>

</html>

**INDEX.JS**

require("dotenv").config();

const express = require("express");

const app = express();

const fileUpload = require("express-fileupload");

app.use(

  fileUpload({

    extended: true,

  })

);

app.use(express.static(\_\_dirname));

app.use(express.json());

const path = require("path");

const ethers = require("ethers");

var port = 3000;

const RPC\_URL = process.env.RPC\_URL;

const PRIVATE\_KEY = process.env.PRIVATE\_KEY;

const CONTRACT\_ADDRESS = process.env.CONTRACT\_ADDRESS;

const { abi } = require("./artifacts/contracts/Voting.sol/Voting.json");

const provider = new ethers.providers.JsonRpcProvider(RPC\_URL);

const signer = new ethers.Wallet(PRIVATE\_KEY, provider);

const contractInstance = new ethers.Contract(CONTRACT\_ADDRESS, abi, signer);

app.get("/", (req, res) => {

  res.sendFile(path.join(\_\_dirname, "index.html"));

});

app.get("/index.html", (req, res) => {

  res.sendFile(path.join(\_\_dirname, "index.html"));

});

app.post("/vote", async (req, res) => {

  var vote = req.body.vote;

  console.log(vote);

  async function storeDataInBlockchain(vote) {

    console.log("Adding the candidate in the voting contract...");

    const tx = await contractInstance.addCandidate(vote);

    await tx.wait();

  }

  const bool = await contractInstance.getVotingStatus();

  if (bool == true) {

    await storeDataInBlockchain(vote);

    res.send("The candidate has been registered in the smart contract");

  } else {

    res.send("Voting is finished");

  }

});

app.listen(port, function () {

  console.log("App is listening on port 3000");

});

**.ENV**

RPC\_URL = "https://rpc-mumbai.maticvigil.com"

PRIVATE\_KEY ="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"

API\_KEY ="ZU892KS2FZ9IMWKKBVJ6RIZXRXK2EBD45Q"

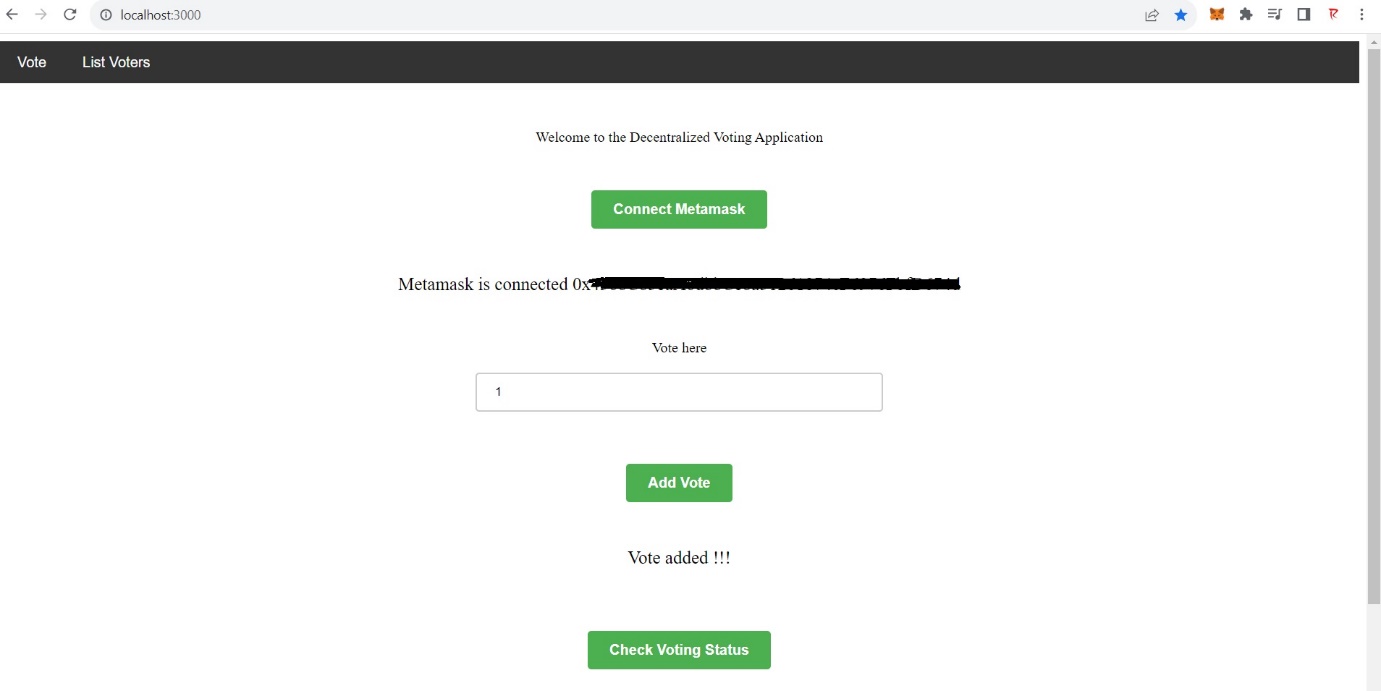
CONTRACT\_ADDRESS  ="0xXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"

**OUTPUT SCREENSHOT:**

**1.ADD YOUR VOTE :**

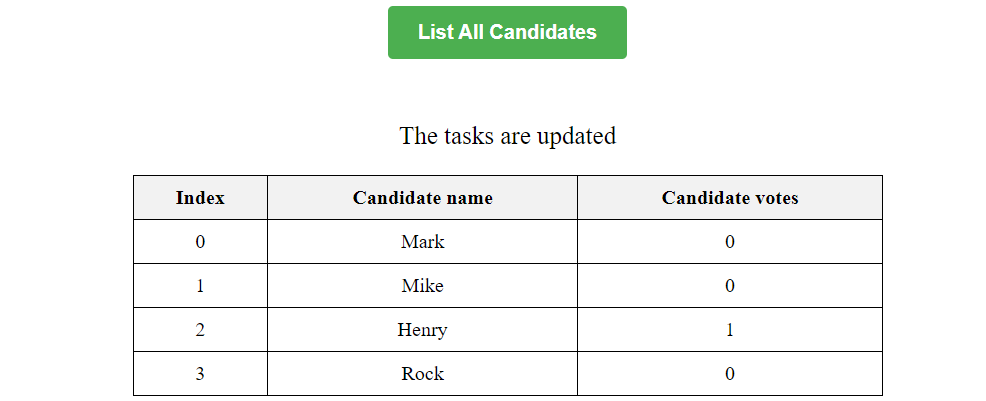
Using “connect metamask” button connect the metamask account and add your vote using your “Add vote” button

Using “check voting status” button , vote status can be viewed

****

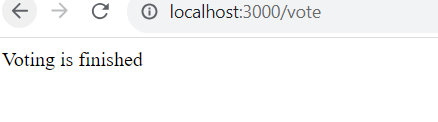
2.  **LIST ALL CANDIDATES:**

We can also view the candidates list and vote count for the candidate



3. **VOTING FINISHED** :

Once the time declared is finished voting will be finished



**CONCLUSION:**

In conclusion, the developed decentralized voting application leverages blockchain technology and the Ethereum platform to create a secure and transparent voting process. By utilizing the Ethereum smart contract, users can cast their votes for various candidates, with the contract maintaining an immutable record of the votes. Metamask integration ensures secure user interactions, and the frontend interface provides real-time information about the voting status, remaining time, and candidate vote counts. This application showcases the potential of blockchain in revolutionizing traditional voting systems by enhancing transparency, security, and trust. However, it's important to continue refining the code, conducting thorough testing, and staying updated with the latest developments in blockchain technology to ensure the continued reliability and functionality of the application.