# Decentralized Voting Web Application

A web app project using Blockchain Technology

In fulfillment of the requirement for the
**Blockchain Technology Course**
Under
**NIELIT Imphal and Globizs Web Solution Pvt. Ltd. Imphal**

By
**Lenin Khangjrakpam**
B.Tech 8th Semester
(MIT, Imphal)

Under the guidance of
**Sir Shengang Maringmei**


राष्ट्रीय इलेक्ट्रॉनिकी एवं सूचना प्रौद्योगिकी संस्थान

National Institute of Electronics & Information Technology, Imphal
An Autonomous Scientific Society under the administrative control of the
Ministry of Electronics & Information Technology (MoE&IT)
Government of India
Akampat P.O.Box-104, Imphal,

# Table of Contents

# Acknowledgment

I have put effort into this project. However, it would not have been possible without your kind support and help. I would like to extend my sincere thanks to all of them.
I would like to express my sincere appreciation to the following individuals and organizations for their support and contributions to this project:

**Project Advisor**

**Mr. Shengang Maringmei:** Blockchain Developer at Globizs, for his invaluable guidance, cooperation, and encouragement during the project period. His expertise and support were indispensable in the completion of this project.

**Institutional Support**

**NIELIT Imphal:** For organizing the training program and providing the platform for learning and growth.

**Globizs Web Solution Pvt. Ltd., Imphal, Manipur:** For their support and resources during the project development phase.

**Training Program Facilitators**

**Miss Swavana Yaikhom:** Faculty at NIELIT Imphal, for her assistance throughout the training program.

**Mr. N. Debachandra Singh:** Director of NIELIT Imphal, for his leadership and vision in fostering educational initiatives.

The **open-source community** for providing invaluable resources and inspiration.

I extend my thanks to all individuals and organizations mentioned above for their contributions to this project. Their support has been instrumental in its successful completion.

# Abstract

This project, titled ***"Decentralized Voting Web Application"*** represents an exploration into leveraging blockchain technology to revolutionize the democratic process. As traditional voting systems face challenges related to transparency, security, and accessibility, blockchain presents a promising solution by offering a decentralized and tamper-resistant platform for conducting elections.

Built primarily using the Solidity programming language for smart contracts, CSS, JavaScript, ReactJS, and Web3.js, the decentralized voting web application aims to provide a secure, transparent, and efficient means for individuals to cast their votes remotely while ensuring the integrity of the electoral process.

The system employs Ethereum smart contracts to record and tally votes in an immutable and transparent manner, thereby mitigating risks associated with fraud and manipulation. Through the utilization of cryptographic techniques and blockchain's decentralized architecture, voter anonymity and ballot secrecy are preserved, fostering trust and confidence in the electoral process.

Key features of the voting web application include voter registration, ballot creation, vote casting, and result verification. Additionally, the platform incorporates user-friendly interfaces to enhance accessibility for voters of all technical proficiencies.

This project contributes to the ongoing discourse on digital democracy and highlights the potential of blockchain technology to redefine electoral systems worldwide. By providing a secure and transparent voting infrastructure, the decentralized voting web application seeks to empower individuals and communities to participate actively in the democratic process, ultimately fostering greater trust and legitimacy in governance.

# Introduction

In an era marked by technological innovation and societal evolution, the traditional methods of conducting elections are facing increasing scrutiny. Issues such as voter fraud, ballot manipulation, and lack of transparency have underscored the need for a more robust and trustworthy electoral process. In response to these challenges, blockchain technology has emerged as a disruptive force with the potential to revolutionize the way we vote.

The "**Decentralized Voting Web Application**" represents a pioneering effort to harness the power of blockchain for democratic governance. By leveraging the principles of decentralization, immutability, and transparency inherent in blockchain technology, this project aims to create a secure and reliable platform for conducting elections in a digital age.

# Objective

The primary objective of this project is to develop a decentralized voting web application that addresses the shortcomings of traditional voting systems while ensuring integrity, security, and accessibility.
Key objectives include:

- **Transparency:** Implementing a transparent and auditable voting process to instill trust and confidence in the electoral outcome.
- **Security:** Utilizing cryptographic techniques and blockchain's decentralized architecture to safeguard against fraud, manipulation, and unauthorized access.
- **Accessibility**: Designing user-friendly interfaces and accessible voting mechanisms using ReactJS to facilitate participation among individuals of diverse backgrounds and technical proficiencies.

## Methodology

The development of the decentralized voting web application is guided by a comprehensive methodology that encompasses:

- **Requirement Analysis:** Identifying the functional and non-functional requirements of the voting system to meet the needs of various stakeholders.
- **Design and Architecture:** Designing the system architecture and user interfaces using ReactJS to ensure scalability, usability, and security.
- **Implementation:** Writing smart contracts in Solidity, integrating with ReactJS for frontend interfaces, and connecting backend components using Web3.js.
- **Testing and Validation:** Conducting rigorous testing procedures to validate the functionality, security, and reliability of the voting application.

## Scope of the Project

The scope of the project encompasses the development of a prototype decentralized voting web application, focusing on key features such as:

- Voter registration and authentication
- Ballot creation and distribution
- Secure and anonymous vote-casting
- Real-time vote tallying and result verification
- Admin functionalities for managing elections and monitoring voting activity

# Technology Used

- Solidity (Ethereum Smart Contracts)
- ReactJS, CSS, Javascript
- web3.js (JS library for interacting with Ethereum smart contract)
- Remix IDE (online solidity IDE for smart contract development)
- Ganache CLI**:** Local Ethereum blockchain for development and testing

# System Architecture

The architecture of the Decentralized Voting Web Application is designed to leverage a combination of blockchain technology, frontend frameworks, and development tools to create a secure, transparent, and user-friendly voting platform.

## Technology Stack

**Backend:**

- **Solidity (Ethereum Smart Contracts):** The core logic of the voting system is implemented through Ethereum smart contracts. These contracts define the rules for voter registration, ballot creation, vote casting, and result tallying.
- **Remix IDE:** An online Solidity IDE used for smart contract development and testing. Remix provides a convenient environment for writing, debugging, and deploying smart contracts to the Ethereum blockchain.

**Frontend:**

- **ReactJS:** A JavaScript library for building user interfaces. ReactJS is used to develop the frontend components of the voting web application, providing a dynamic and responsive user experience.
- **CSS:** Cascading Style Sheets are utilized for styling the frontend interfaces, ensuring consistency and aesthetic appeal.
- **JavaScript:** Alongside ReactJS, JavaScript is employed to implement interactive features and functionality within the voting application.

**Blockchain Integration:**

- **web3.js:** A JavaScript library that allows interaction with the Ethereum blockchain. web3.js facilitates communication between the frontend interfaces and the Ethereum smart contracts, enabling users to interact with the voting system seamlessly.

**Development Tools:**

- **Ganache CLI**: A local Ethereum blockchain for development and testing purposes. Ganache CLI provides a simulated blockchain environment where smart contracts can be deployed and tested without incurring transaction costs or waiting for block confirmations on the main Ethereum network.

## System Components

The Decentralized Voting Web Application comprises the following components:

- **Smart Contracts:** Implemented in Solidity, these Ethereum smart contracts define the core logic and rules of the voting system, including voter registration, ballot creation, and vote tallying.

- **Frontend Interfaces:** Developed using ReactJS, CSS, and JavaScript, the frontend interfaces provide users with an intuitive and interactive voting experience. Users can register, cast votes, and view election results through these interfaces.
- **web3.js Integration**: web3.js facilitates communication between the frontend interfaces and the Ethereum blockchain. It enables users to interact with the smart contracts deployed on the blockchain, such as submitting transactions and retrieving voting data.
- **Remix IDE:** Utilized for smart contract development, Remix IDE offers an online environment for writing, testing, and deploying Solidity smart contracts. It provides developers with tools for debugging and analyzing contract behavior.
- **Ganache CLI:** Used as a local Ethereum blockchain for testing and development purposes, Ganache CLI allows developers to deploy and interact with smart contracts in a simulated blockchain environment. It enables rapid iteration and debugging of contract functionality without incurring transaction costs.




\

# Implementation

Complete source code for the project can be found at:
https://github.com/LeninKhangjrakpam/Voting-DAPP.git

## System Overview

**Smart Contract Design:**

The smart contract architecture for the Decentralized Voting Web Application is meticulously designed to ensure the integrity, transparency, and security of the voting process. It encompasses the following components:

- **Voter Registration:** Users are registered on the blockchain through the Voter struct, storing essential information such as voter ID, Ethereum account address, name, and the address of the registrar.
- **Candidate Registration:** Candidates are registered on the blockchain using the Candidate struct, which includes candidate ID, voter ID, address, and party affiliation.
- **Party Registration:** Political parties are registered on the blockchain via the Party struct, containing party ID and name.
- **Vote Event Setup:** Election administrators create voting events using the VoteEvent struct, specifying event ID, name, start date, duration, creator's address, and associated candidate addresses.
- **Vote Casting:** Registered voters cast their votes through the Poll Page interface, selecting voting events, and candidates, and inputting their voter ID. Votes are recorded on the blockchain and associated with the respective voter and candidate.

**Frontend Interface:**

The frontend interface of the Decentralized Voting Web Application provides a seamless and intuitive user experience for both voters and administrators. It consists of the following pages:

- **Dashboard:** The Dashboard page displays live vote results to any user, providing real-time updates on ongoing voting events and their respective outcomes.
- **Information Page:** The Information Page presents users with details about currently active voting events and past voting events, enabling them to stay informed about the electoral process.
- **Poll Page:** The Poll Page serves as the primary interface for users to cast their votes. Users select voting events from a dropdown menu, choose their preferred candidates, and input their voter ID to complete the voting process.

- **Candidate Registration:** Administrators utilize the Candidate Registration page to register candidates for upcoming elections, providing necessary information such as voter ID, address, and party affiliation. Users can also do this by itself by submitting their voter ID and party name
- **Voter Registration:** Administrators manage voter registration through the Voter Registration page, enabling eligible individuals to register as voters by providing their personal details and obtaining a voter ID.
- **Party Registration:** Administrators facilitate party registration via the Party Registration page, allowing political organizations to register their party names and obtain unique party IDs.
- **Vote Event Registration:** Only administrators have access to the Vote Event Registration page, where they create new voting events by specifying event details such as name, start date, and duration.

**Interaction Flow:**

- **User Interaction:** In order for users to participate in vote events, they need to first register on the voter page and retrieve their unique ID. Then, users can poll or vote in the voting event by going to the "*poll/"* page selecting voting events, choosing candidates, and casting their votes securely on the blockchain.
- **Admin Interaction:** Administrators utilize various frontend pages to manage the electoral process, including registering candidates, voters, parties, and vote events. Admin and voter can monitor voting activity and view live vote results through the Dashboard page.
- **Blockchain Interaction:** Both users and administrators interact with the smart contract deployed on the Ethereum blockchain through the frontend interface. Transactions, such as voter registration, candidate registration, and vote casting, are securely recorded on the blockchain, ensuring transparency and immutability.

# Conclusion

The development of the Decentralized Voting Web Application marks a significant milestone in leveraging blockchain technology to enhance the democratic process. Through meticulous planning, innovative design, and collaborative efforts, this project has successfully achieved its objectives of creating a transparent, secure, and accessible platform for conducting elections in a digital age.

**Key Achievements**

- **Enhanced Transparency:** By storing voting records on the Ethereum blockchain, the Decentralized Voting Web Application ensures transparency and immutability of the electoral process. Users can verify the integrity of election results, fostering trust and confidence in the democratic process.
- **Secure Voting Mechanism:** Utilizing cryptographic techniques and decentralized architecture, the voting platform offers a secure and tamper-resistant environment for casting votes. The integration of smart contracts ensures that votes are recorded accurately and cannot be altered or manipulated.
- **User-Friendly Interface:** The frontend interfaces of the voting application are designed to be intuitive and user-friendly, enabling voters to participate in elections effortlessly. Features such as the Poll Page and Dashboard provide users with a seamless voting experience and real-time access to election results.
- **Administrative Controls:** Election administrators have access to robust administrative controls, allowing them to manage voter registration, candidate registration, party registration, and vote event setup. The system provides administrators with the tools and insights needed to oversee the entire electoral process effectively.

**Future Directions**

While the Decentralized Voting Web Application has achieved significant success, there are opportunities for further enhancement and expansion in the future:

- **Scalability:** Exploring solutions for scaling the voting platform to accommodate a larger user base and handle increased transaction volume.
- **Accessibility:** Implementing features to enhance accessibility for voters with disabilities and those with limited internet connectivity.
- **Integration with Identity Solutions:** Investigating the integration of decentralized identity solutions to improve voter authentication and reduce the risk of identity fraud.

- **International Adoption**: Exploring opportunities to deploy the voting platform in international contexts, collaborating with governments and organizations to promote democratic principles globally.

# Reference


- Solidity Documentation: https://soliditylang.org/
- Remix Ethereum IDE: https://remix.ethereum.org/
- web3JS Documentation: https://web3js.readthedocs.io/
- ReactJS: https://react.dev/
- CSS: https://developer.mozilla.org/

# User Interface Samples

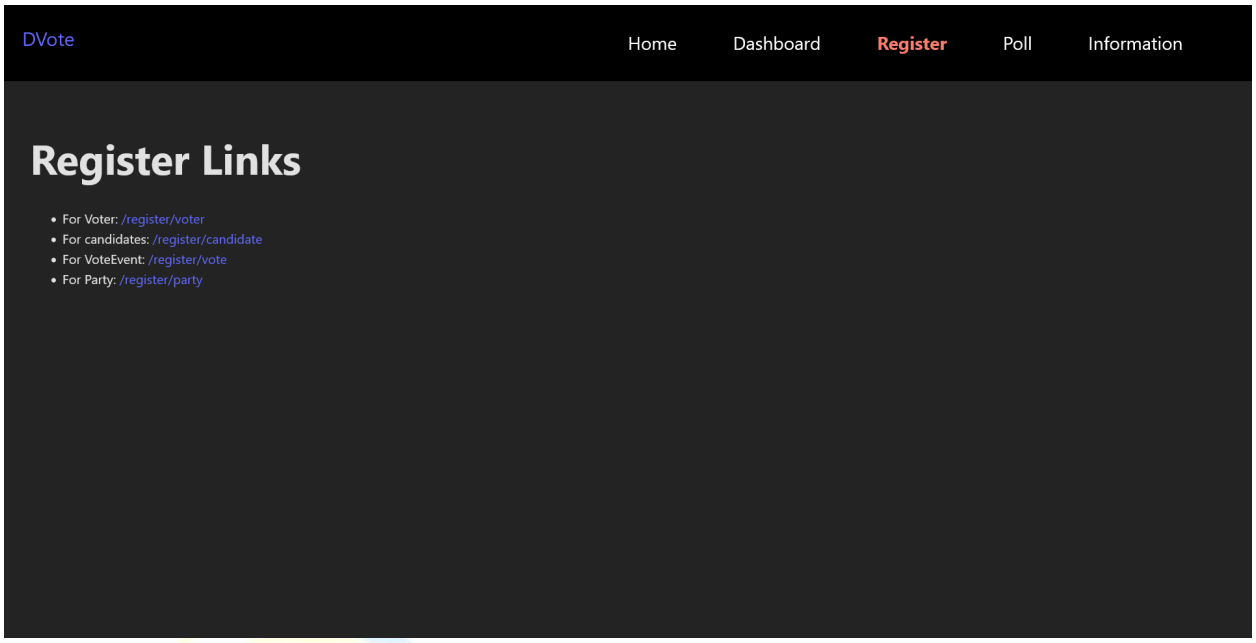



**Figure:** Registration Link Page

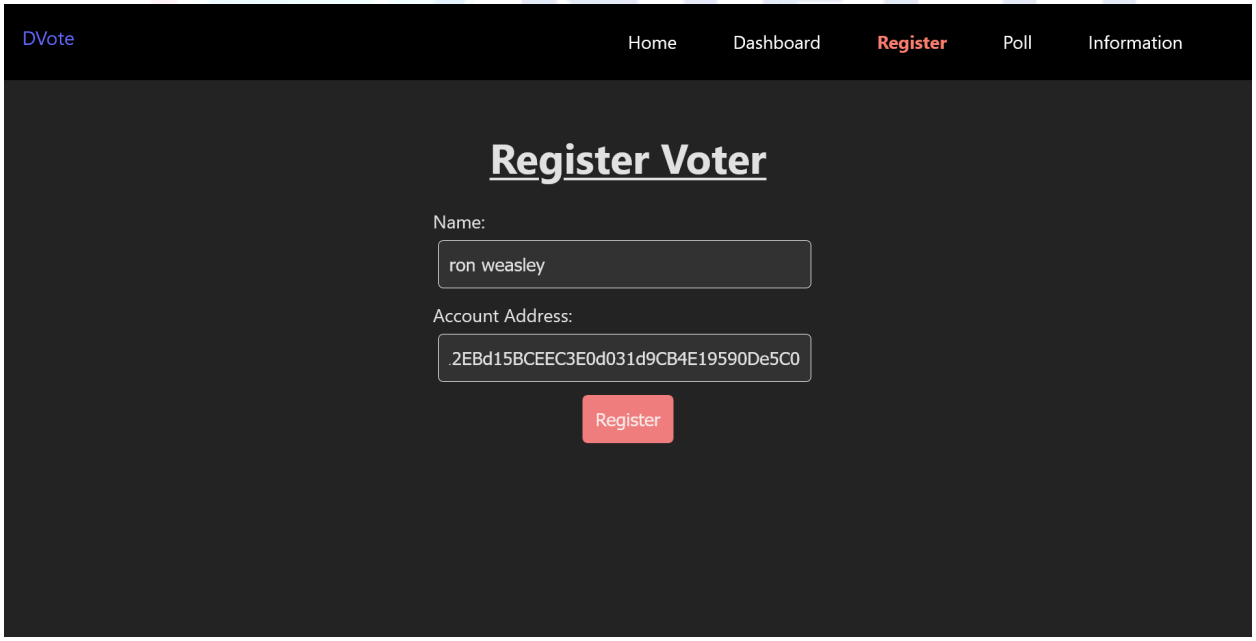



**Figure:** Voter Registration Page

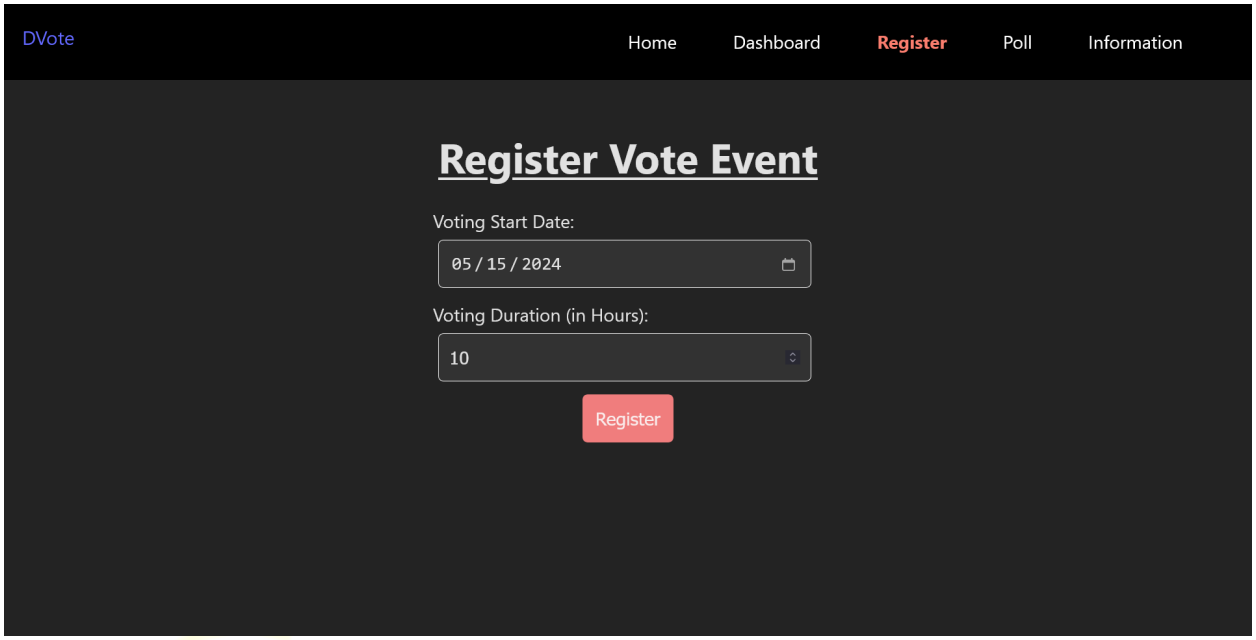



**Figure:** Vote Event Registration Page

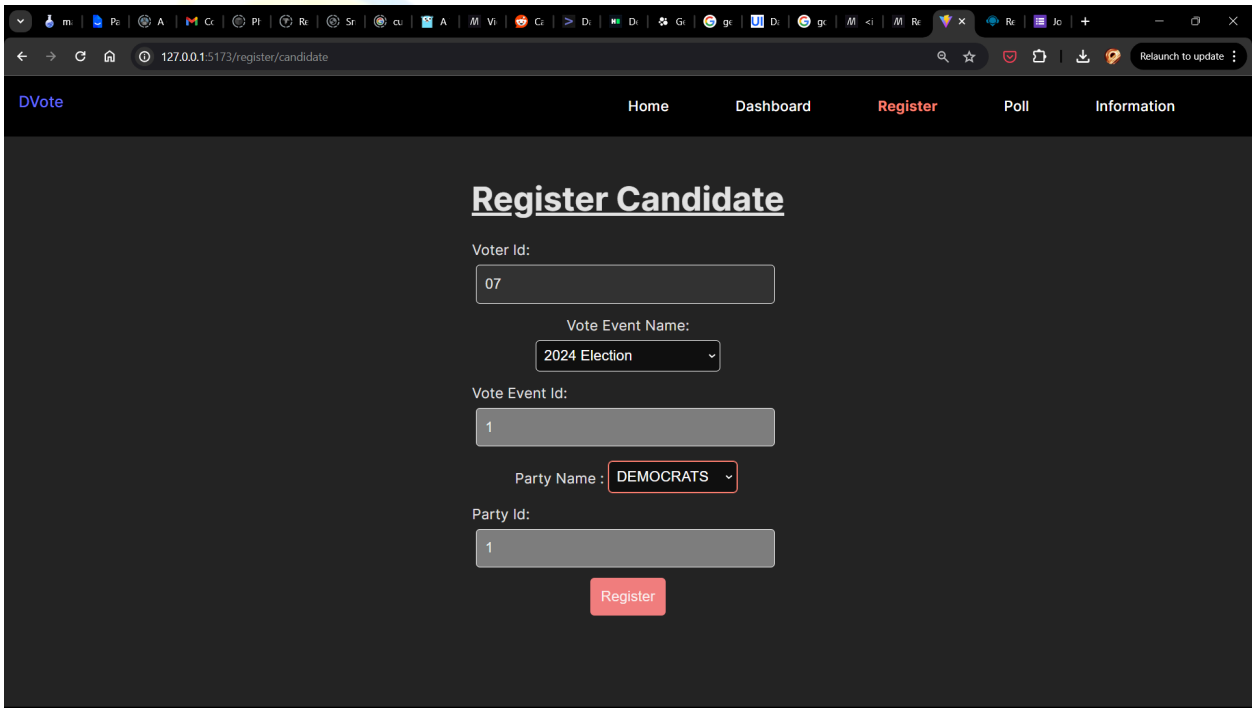



**Figure:** Candidate Registration Page

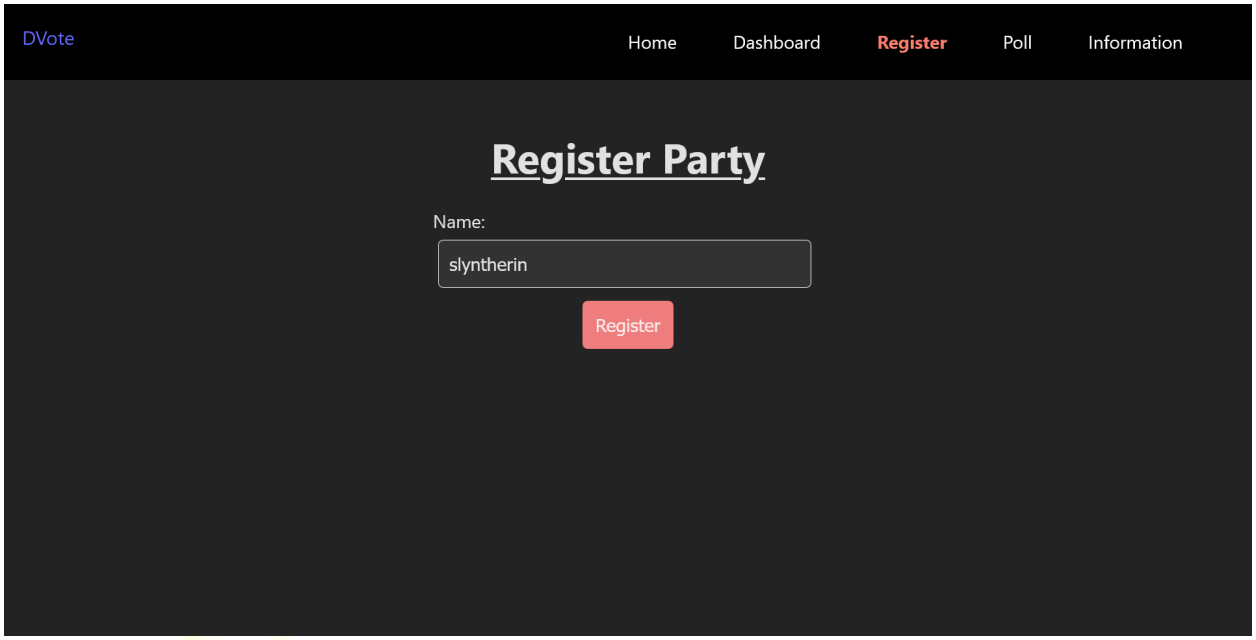
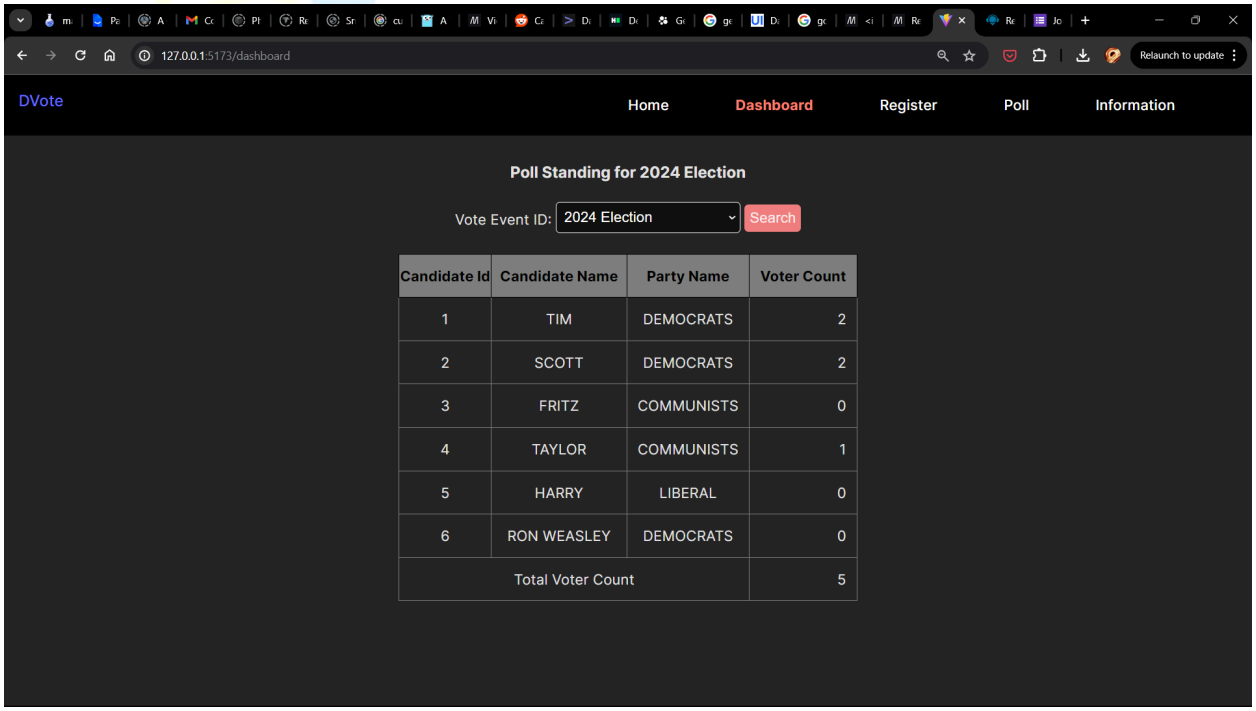



**Figure:** Party Registration Page




| Candidate Id | Candidate Name | Party Name | Voter Count |
|---|---|---|---|
| 1 | TIM | DEMOCRATS | 2 |
| 2 | SCOTT | DEMOCRATS | 2 |
| 3 | FRITZ | COMMUNISTS | 0 |
| 4 | TAYLOR | COMMUNISTS | 1 |
| 5 | HARRY | LIBERAL | 0 |
| 6 | RON WEASLEY | DEMOCRATS | 0 |
| Total Voter Count | | | 5 |

**Figure:** Dashboard Page

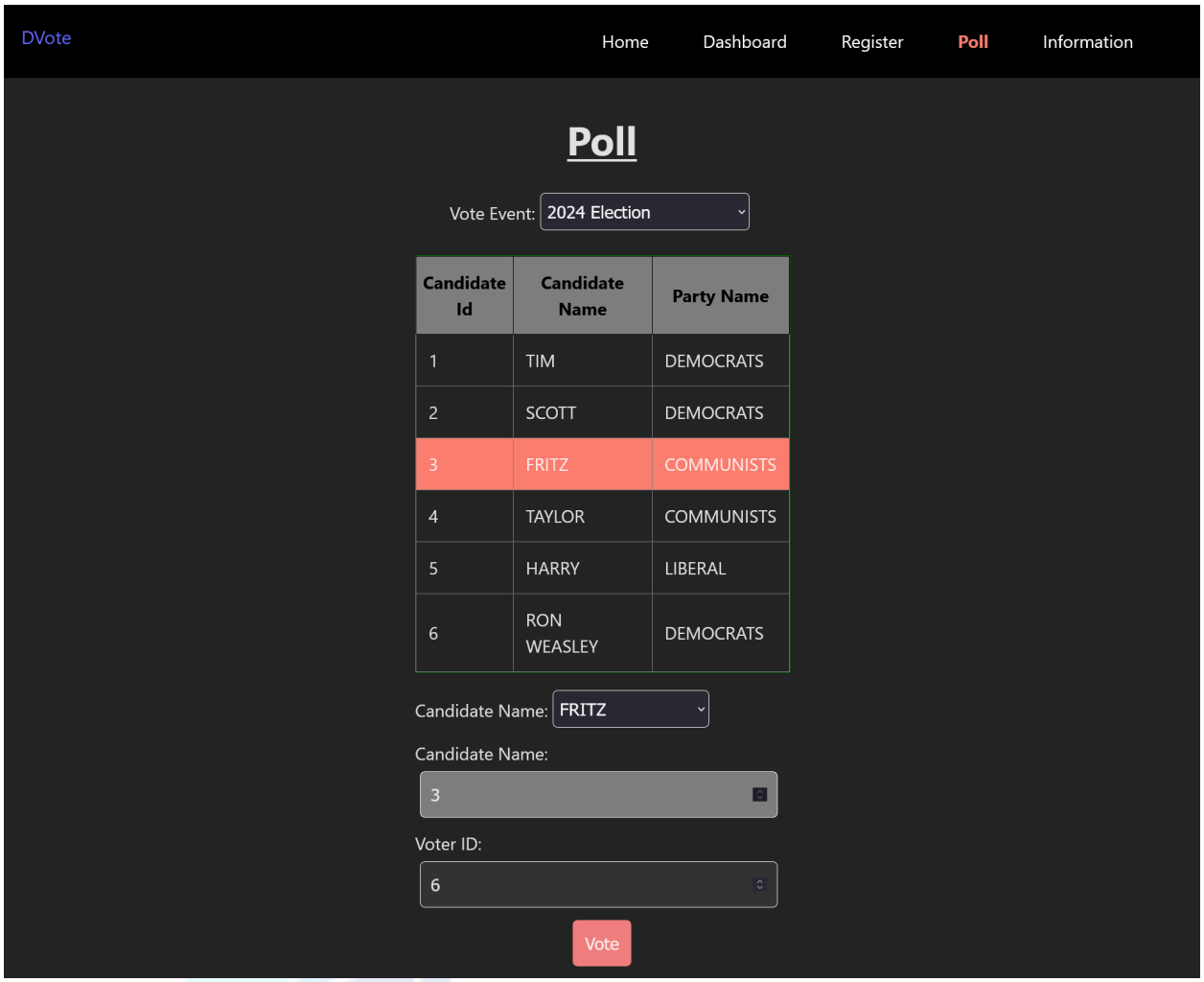



**Figure:** Poll or Voting Page

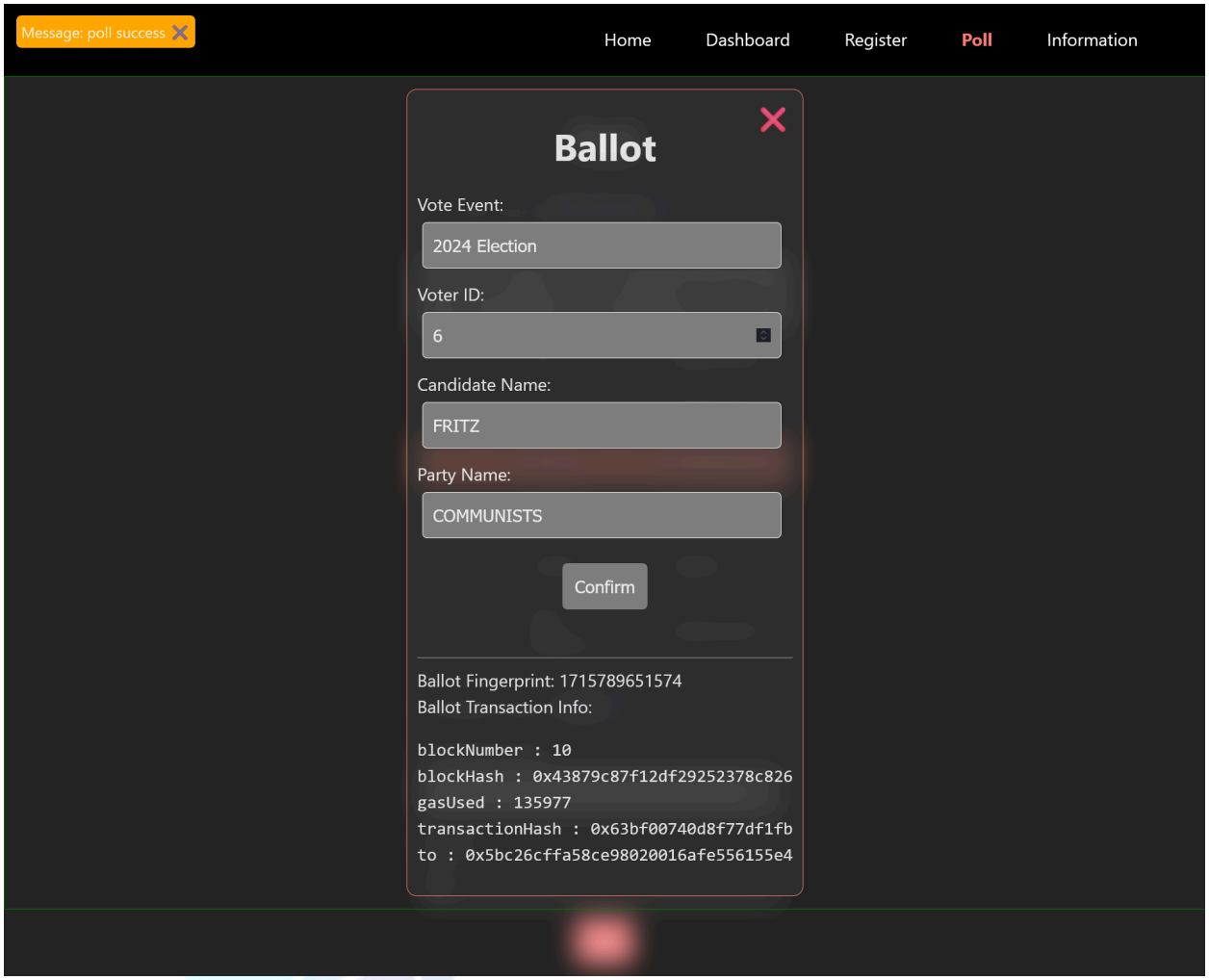



**Figure:** Ballot Generation Modal (successful vote, as shown in the message box (*Poll Success*))

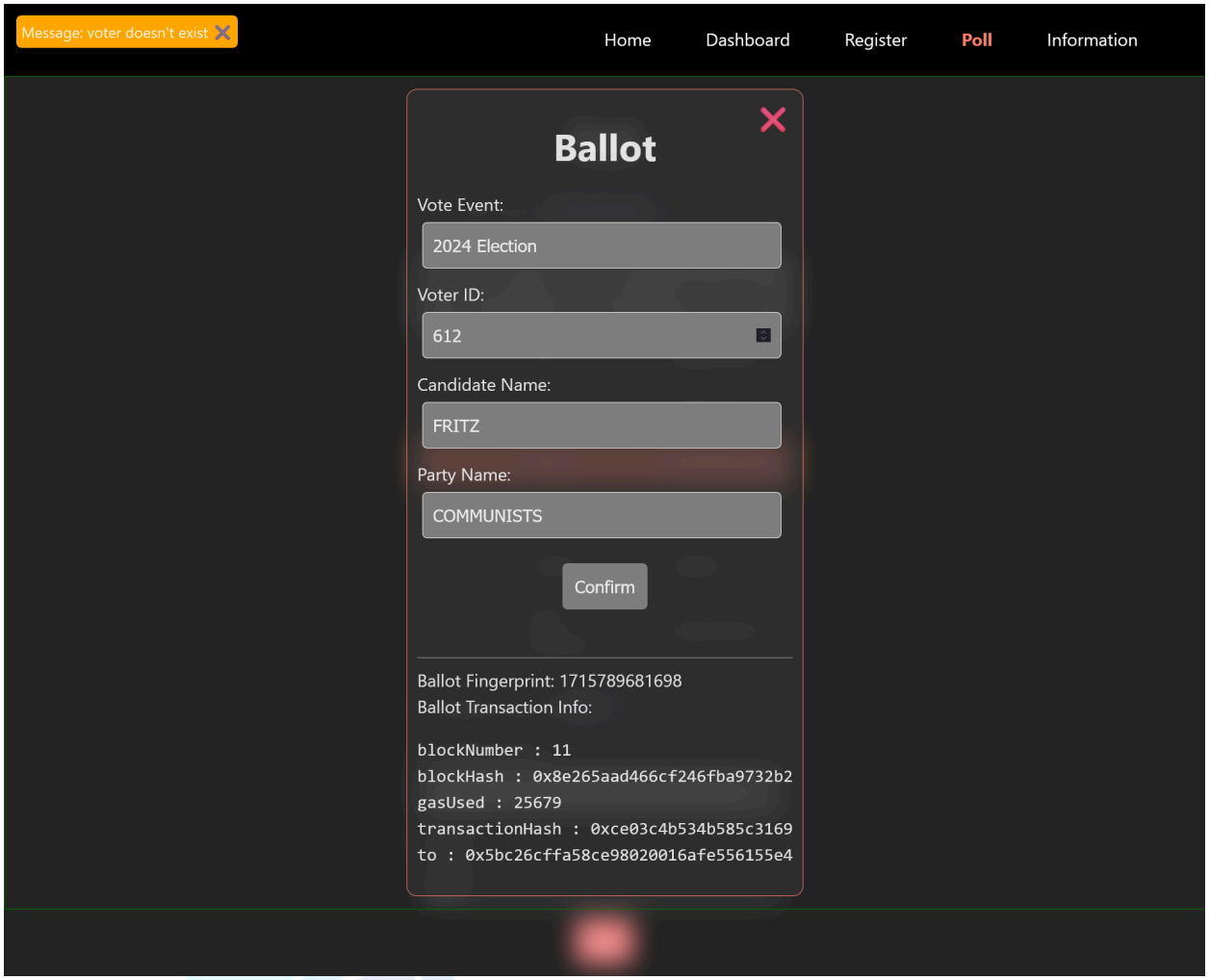



**Figure:** Ballot Generation Modal (unsuccessful vote, as shown in the message box (*Voter ID doesn't exist*))

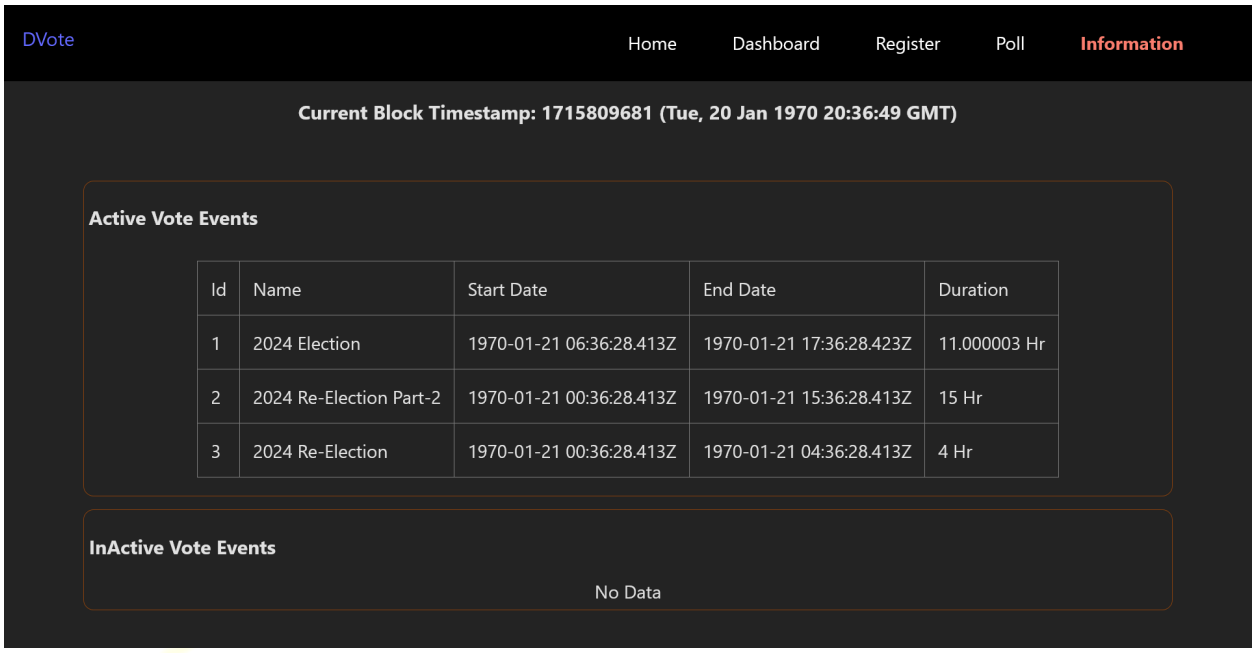



**Figure:** Information Page (**view active, past, inactive, upcoming vote events**)

# Appendix

**Solidity Code for Vote Smart Contract**

Source: https://gist.github.com/LeninKhangjrakpam/f05bc8fbf0edb935f851019a5a8c4480

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.7.6;

pragma abicoder v2;

contract DVote {
    struct Voter {
        uint256 voterId;
        address addr;
        string name;
        address registerBy;
    }

    struct Candidate {
        uint256 id;
        uint256 voterId;
        address addr;
        uint256 partyId;
    }

    struct VoteEvent {
        uint256 id;
        string name;
        uint256 startDate;
        uint256 duration;
        address createdBy;
        address[] candidateAddr;
    }

    struct Party {
        uint256 id;
        string name;
    }

    mapping(uint256 => Voter) public VoterMap;
    mapping(uint256 => Party) public PartyMap;
    mapping(uint256 => VoteEvent) public VoteEventMap;
```

```
    mapping(uint256 => Candidate[]) public CandidateMap; // VoteEvent =>
Candidate[]
    mapping(uint256 => mapping(uint256 => uint256[])) public EvntCndVtr; //
VoteEvent => (CandidateId => VoterId[])

    uint256 voterCount = 0;
    uint256 voteEventCount = 0;
    uint256 partyCount = 0;

    address[] adminAddrs;

    constructor() {
        adminAddrs.push(msg.sender);
        init();
    }

    function isAdmin(address addr) public view returns (bool) {
        for (uint256 i = 0; i < adminAddrs.length; i++) {
            if (adminAddrs[i] == addr) {
                return true;
            }
        }
        return false;
    }

    function addAdmin(address addr) public {
        require(isAdmin(msg.sender), "admin can be add by admin user
only");
        adminAddrs.push(addr);
    }

    function registerVoter(address addr, string memory name) public {
        require(isAdmin(msg.sender), "voter can be add only by admin");

        // TODO: Check voter already register by account address
        Voter memory v = Voter({
            voterId: voterCount + 1,
            addr: addr,
            name: name,
            registerBy: msg.sender
        });
```

```
        VoterMap[voterCount + 1] = v;
        voterCount++;
    }

    function isVoterExist(uint256 voterId) private view returns (bool) {
        return VoterMap[voterId].voterId != 0;
    }

    function isVoteEvntExist(uint256 voteEventId) private view returns
(bool) {
        return VoteEventMap[voteEventId].id != 0;
    }

    function isPartyExist(uint256 partyId) private view returns (bool) {
        return PartyMap[partyId].id != 0;
    }

    function createParty(string memory partyName) public {
        for (uint256 i = 0; i < partyCount; i++) {
            if (
                keccak256(bytes(PartyMap[i + 1].name)) ==
                keccak256(bytes(partyName))
            ) {
                revert(
                    "new party cannot be created with same name as existing
party name"
                );
            }
        }
        Party memory p = Party({id: partyCount + 1, name: partyName});

        PartyMap[partyCount + 1] = p;
        partyCount++;
    }

    function createVoteEvnt(
        string memory eventName,
        uint256 startDate,
        uint256 duration
    ) public {
        require(isAdmin(msg.sender), "admin can only create vote event");

        address[] memory cd;
```

```
        VoteEvent memory ve = VoteEvent({
            id: voteEventCount + 1,
            name: eventName,
            startDate: startDate,
            duration: duration,
            candidateAddr: cd,
            createdBy: msg.sender
        });

        VoteEventMap[voteEventCount + 1] = ve;
        voteEventCount++;
    }

    function isCandidateExist(uint256 voterId, uint256 voteEvntId)
        public
        view
        returns (bool)
    {
        for (uint256 i = 0; i < CandidateMap[voteEvntId].length; i++) {
            if (CandidateMap[voteEvntId][i].voterId == voterId) {
                return true;
            }
        }
        return false;
    }

    function registerCandidate(
        uint256 voterId,
        uint256 voteEvntId,
        uint256 partyId
    ) public {
        require(isVoterExist(voterId) == true, "Voter doesn't exist");
        require(
            isVoteEvntExist(voteEvntId) == true,
            "candidate cannot register for non-existing vote evnt"
        );
        require(
            VoteEventMap[voteEvntId].startDate > block.timestamp,
            "candidate cannot register voting already started"
        );
        require(
            isPartyExist(partyId) == true,
            "candidate cannot registrer for non existing party"
```

```
    );
    require(
        isCandidateExist(voterId, voteEvntId) == false,
        "candidate has already register for this vote event"
    );

    address addr = VoterMap[voterId].addr;
    uint256 candCount = VoteEventMap[voteEvntId].candidateAddr.length;

    Candidate memory cd = Candidate({
        id: candCount + 1,
        voterId: voterId,
        addr: addr,
        partyId: partyId
    });

    VoteEventMap[voteEvntId].candidateAddr.push(addr);
    CandidateMap[voteEvntId].push(cd);
    // EvntCndVtr[voteEvntId] = new mapping(uint => uint[]);
    EvntCndVtr[voteEvntId][voterId] = new uint256[](0);
}

function isCndExist(uint256 voteEvntId, uint256 cndId)
    public
    view
    returns (bool)
{
    for (uint256 i = 0; i < CandidateMap[voteEvntId].length; i++) {
        if (CandidateMap[voteEvntId][i].id == cndId) {
            return true;
        }
    }
    return false;
}

// If a voter already voted, removed it from poll, otherwise do nothing
function voterRemove(uint256 voterId, uint256 voteEvnt) public {
    for (uint256 i = 0; i < CandidateMap[voteEvnt].length; i++) {
        uint256 cdId = CandidateMap[voteEvnt][i].voterId;
        uint256 len = EvntCndVtr[voteEvnt][cdId].length;
        for (uint256 j = 0; j < len; j++) {
            if (EvntCndVtr[voteEvnt][cdId][j] == voterId) {
                // remove elm from array
```

```
                    EvntCndVtr[voteEvnt][cdId][j] =
EvntCndVtr[voteEvnt][cdId][
                        len - 1
                    ];
                    EvntCndVtr[voteEvnt][cdId].pop();
                    return;
                }
            }
        }
    }

    function getCndVId(uint256 cndId, uint256 voteEvnt)
        public
        view
        returns (uint256)
    {
        Candidate[] memory c = CandidateMap[voteEvnt];
        for (uint256 i = 0; i < c.length; i++) {
            if (c[i].id == cndId) {
                return c[i].voterId;
            }
        }
        return 0;
    }

    event ReturnedValue(string retStr);

    function poll(
        uint256 voterId,
        uint256 cndId,
        uint256 voteEvntId
    ) public returns (string memory) {
        if (isVoterExist(voterId) == false) {
            emit ReturnedValue("voter doesn't exist");
            return "voter doesn't exist";
        }
        if (isVoteEvntExist(voteEvntId) == false) {
            emit ReturnedValue("voter cannot vote for non existing vote
event");
            return "voter cannot vote for non existing vote event";
        }
        if (isCndExist(voteEvntId, cndId) == false) {
            emit ReturnedValue("candidate doesnt exist in given
```

```
voteEvent");
            return "candidate doesnt exist in given voteEvent";
        }

        // TODO: require (check voting event started)
        // Check Vote isnt over yet
        if (
            VoteEventMap[voteEvntId].startDate +
                VoteEventMap[voteEvntId].duration <=
            block.timestamp
        ) {
            emit ReturnedValue("voter cannot vote after vote event end");
            return "voter cannot vote after vote event end";
        }

        // Check voter already voted
        voterRemove(voterId, voteEvntId);
        // Poll Vote
        uint256 cdVId = getCndVId(cndId, voteEvntId);
        if (cdVId == 0) {
            emit ReturnedValue("candidate voterid cannot be 0");
            return "candidate voterid cannot be 0";
        }
        EvntCndVtr[voteEvntId][cdVId].push(voterId);

        emit ReturnedValue("poll success");
        return "poll success";
    }

    function getVoterCount(uint256 cndId, uint256 voteEvntId)
        public
        view
        returns (uint256)
    {
        // Return voter count for a give cndId
        require(
            isVoteEvntExist(voteEvntId) == true,
            "votercannot vote for non existing vote event"
        );
        require(
            isCndExist(voteEvntId, cndId) == true,
            "candidate doesnt exist in given voteEvent"
        );
```

```
    return EvntCndVtr[voteEvntId][cndId].length;
}

struct VoteE {
    uint256 id;
    string name;
    uint256 startDate;
    uint256 duration;
}

// Get liost of vote events
function getVoteEvents() public view returns (VoteE[] memory) {
    VoteE[] memory ve = new VoteE[](voteEventCount);
    for (uint256 i = 0; i < voteEventCount; i++) {
        VoteEvent memory v = VoteEventMap[i + 1];
        ve[i] = VoteE({
            id: v.id,
            name: v.name,
            startDate: v.startDate,
            duration: v.duration
        });
    }
    return ve;
}


struct CandidateInfo {
    uint256 candId;
    uint256 voterId;
    string name;
    address addr;
    uint256 partyId;
    string partyName;
}

// Get all candidate info for a given vote event
function getCandidateInfo(uint256 voteEvntId)
    public
    view
    returns (CandidateInfo[] memory)
{
    require(
```

```
        isVoteEvntExist(voteEvntId),
        "given voting event id event does not exist"
    );

    uint256 candLen = CandidateMap[voteEvntId].length;
    CandidateInfo[] memory c = new CandidateInfo[](candLen);
    for (uint256 i = 0; i < candLen; i++) {
        uint256 cndId = CandidateMap[voteEvntId][i].id;
        address addr = CandidateMap[voteEvntId][i].addr;
        uint256 voterId = CandidateMap[voteEvntId][i].voterId;
        uint256 partyId = CandidateMap[voteEvntId][i].partyId;

        c[i] = CandidateInfo({
            candId: cndId,
            voterId: voterId,
            name: VoterMap[voterId].name,
            addr: addr,
            partyId: partyId,
            partyName: PartyMap[partyId].name
        });
    }
    return c;
}

// Getter for voteEventmap
function getVoteEvntInfo(uint256 voteEvntId)
    public
    view
    returns (VoteEvent memory)
{
    require(
        isVoteEvntExist(voteEvntId) == true,
        "voteEvnt Id doesnt exist"
    );
    return VoteEventMap[voteEvntId];
}

struct VoteInfo {
    uint256 cndId;
    uint256 voteCount;
    string name;
    uint256 partyId;
    string partyName;
```

```
    }

    // Return poll standing for a given vote event
    function getVoteInfo(uint256 voteEvntId)
        public
        view
        returns (VoteInfo[] memory)
    {
        // Check voteEvnt exist
        require(
            isVoteEvntExist(voteEvntId) == true,
            "voteEvnt Id doesnt exist"
        );

        uint256 len = CandidateMap[voteEvntId].length;
        VoteInfo[] memory vi = new VoteInfo[](len);
        for (uint256 i = 0; i < len; i++) {
            // Loop through all candidates
            uint256 cndId = CandidateMap[voteEvntId][i].id;
            uint256 partyId = CandidateMap[voteEvntId][i].partyId;
            uint256 voteCount = EvntCndVtr[voteEvntId][cndId].length;
            vi[i] = VoteInfo({
                cndId: cndId,
                voteCount: voteCount,
                name: VoterMap[CandidateMap[voteEvntId][i].voterId].name,
                partyId: partyId,
                partyName: PartyMap[partyId].name
            });
        }
        return vi;
    }

    bool initCalled = false;

    // Fill up so,e user for testing purpose
    function init() public {
        if (!initCalled) {
            // Add voters
            registerVoter(0x5B38Da6a701c568545dCfcB03FcB875f56beddC4,
"tim");
            registerVoter(0x5B38Da6a701c568545dCfcB03FcB875f56beddC4,
"scott");
            registerVoter(0x5B38Da6a701c568545dCfcB03FcB875f56beddC4,
```

```
"fritz");
            registerVoter(0x5B38Da6a701c568545dCfcB03FcB875f56beddC4,
"taylor");
            registerVoter(0x5B38Da6a701c568545dCfcB03FcB875f56beddC4,
"harry");
            registerVoter(
                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4,
                "hermione"
            );
            // Add VoteEvent
            uint256 conv = 1000 * 60 * 60;
            createVoteEvnt(
                "2024 Election",
                block.timestamp + 10 * conv,
                10 + 11 * conv
            );
            createVoteEvnt(
                "2024 Re-Election Part-2",
                block.timestamp + 4 * conv,
                15 * conv
            );
            createVoteEvnt(
                "2024 Re-Election",
                block.timestamp + 4 * conv,
                4 * conv
            );
            // Add Party
            createParty("democrats");
            createParty("communists");
            createParty("liberal");
            // Add candidate
            registerCandidate(1, 1, 1);
            registerCandidate(2, 1, 1);
            registerCandidate(3, 1, 2);
            registerCandidate(4, 1, 2);
            registerCandidate(5, 1, 3);

            // Poll vote
            poll(1, 1, 1);
            poll(2, 1, 1);
            poll(3, 2, 1);
            poll(4, 2, 1);
            poll(5, 4, 1);
```

```
        } else {
            revert("init function already called !");
        }
        initCalled = true;
    }

    function getBlockTimestmp() public view returns (uint256) {
        return block.timestamp;
    }

    function getPartyLists() public view returns (Party[] memory) {
        Party[] memory p = new Party[](partyCount);
        for (uint256 i = 0; i < partyCount; i++) {
            p[i] = PartyMap[i + 1];
        }
        return p;
    }
}
```