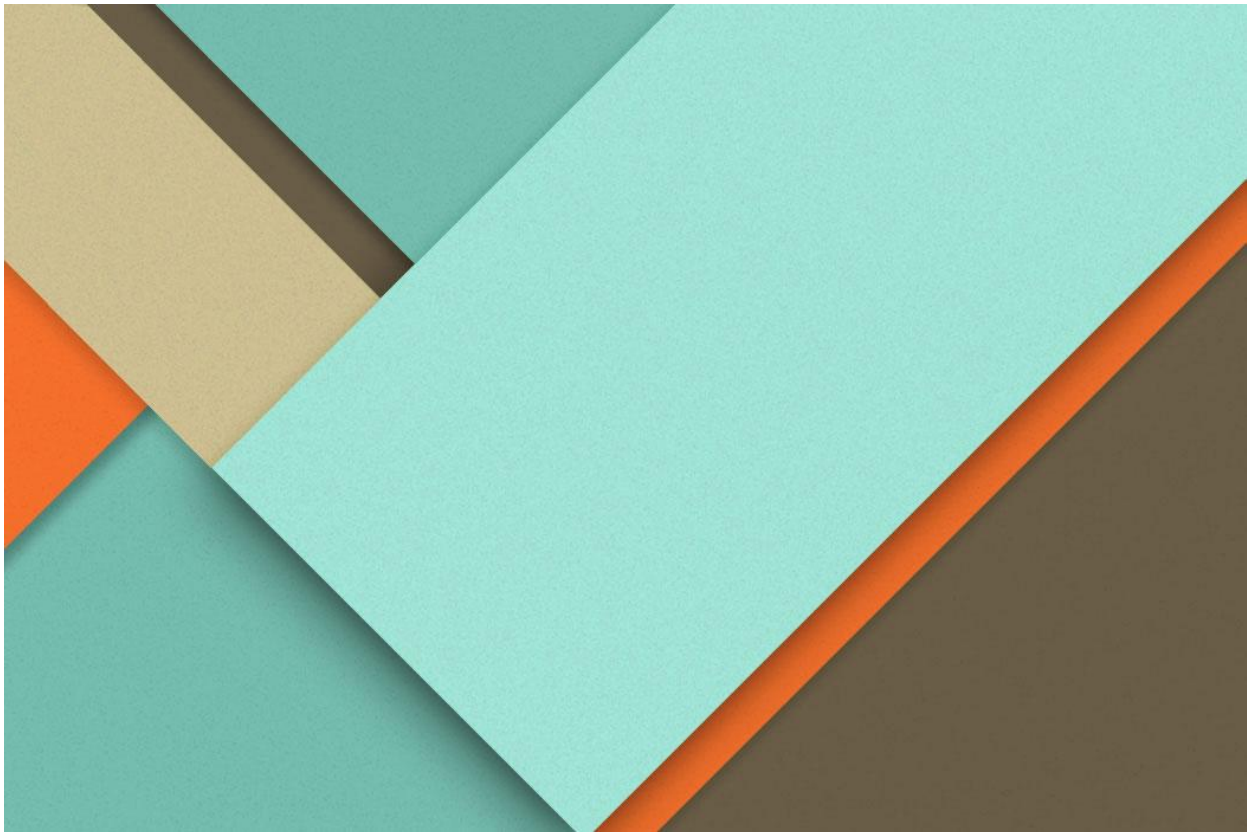


Secure and Trustless Online Voting Platform with Blockchain Technology

(using React and Solidity)

CED-VB10 - GitHub Repo

Hashirahmed K B

ahmedhashir96@gmail.com

+91 9633078242

Overview

This application is a blockchain-based online voting platform that utilizes the Ethereum blockchain to create a tamper-proof ledger of votes, increasing transparency, reducing the potential for voter fraud, and improving public trust in the democratic process.

The system is built using Solidity and React, and allows eligible voters to cast their vote securely and easily from anywhere in the world. By leveraging the benefits of blockchain technology, the system has the potential to revolutionize the way voting is conducted not only in traditional elections but also in corporate board meetings, academic committee elections, and community initiatives.

The significance of this application lies in the fact that the traditional voting process is fraught with challenges that can undermine the legitimacy of election results and reduce public trust in the democratic process. This system provides a fast, efficient, and low-cost alternative to traditional voting systems, while also ensuring the privacy and security of voters' information. By using smart contracts to automate the counting and verification of ballots, the proposed system creates a decentralized, trustless voting process that is transparent and auditable, thus improving the integrity and trustworthiness of elections.

Problem Statement

The traditional voting process is fraught with challenges that can undermine the legitimacy of election results and reduce public trust in the democratic process. Some of the key issues with traditional voting systems include the potential for voter fraud, lack of transparency, and high costs. These problems can lead to questions about the accuracy of election results, which can erode public confidence in the democratic process.

To address these challenges, what I propose is a blockchain-based voting system that utilizes the transparency and security of the Ethereum blockchain to create a tamper-proof ledger of votes. By using smart contracts to automate the counting and verification of ballots, we can create a decentralized, trustless voting process that is transparent and auditable. This system will reduce the potential for voter fraud, increase transparency, and lower costs compared to traditional voting systems.

The proposed system will be built using Solidity and React, and will allow eligible voters to cast their vote securely and easily from anywhere in the world. The system will provide a fast, efficient, and low-cost alternative to traditional voting systems, while also ensuring the accuracy and legitimacy of election results. By building a blockchain-based voting system, we can improve public trust in the democratic process and create a more secure, transparent, and trustworthy voting system for all.

Proposed System

The proposed system is a blockchain-based voting application built using Solidity and React. It utilizes the transparency and security of the Ethereum blockchain to create a tamper-proof ledger of votes, which ensures the accuracy and legitimacy of election results. By using smart contracts to automate the counting and verification of ballots, we can create a decentralized, trustless voting process that is transparent and auditable.

This system provides a fast, efficient, and low-cost alternative to traditional voting systems, while also ensuring the privacy and security of voters' information. The use of blockchain technology allows us to create a system that is resistant to tampering and fraud, while also providing a high degree of transparency and accountability.

The system that I'm proposing has the potential to greatly improve the integrity and trustworthiness of elections, while also providing a more efficient and cost-effective solution compared to traditional voting systems. By leveraging the benefits of blockchain technology, we can create a more secure, transparent, and trustworthy voting system that can increase public trust in the democratic process.

This blockchain-based voting system has the potential to revolutionize the way voting is conducted in a variety of contexts beyond the scope of our course project. For example, the system could be used to improve the voting processes in corporate board meetings, academic committee elections, and community initiatives.

In the corporate world, This system could provide a transparent and secure way for shareholders to cast their votes and ensure that the election process is fair and free from fraud. This could help increase shareholder trust in the company and lead to better decision-making overall.

In the academic world, our system could be used to conduct student council and faculty elections, ensuring that the election process is transparent and fair. This could help increase student and faculty engagement in the election process and improve the overall governance of the institution.

In community initiatives, the system could be used to conduct local government elections and community polls, ensuring that the voice of the people is heard and the election process is free from fraud. This could help increase community participation and engagement in the democratic process, leading to more inclusive and equitable decision-making.

Overall, my blockchain-based voting system has the potential to provide a secure, transparent, and cost-effective solution for a variety of voting applications, leading to a more fair and equitable society.

Existing System

The existing voting system is centralized, and there is no tamper-proof way to ensure the accuracy and legitimacy of election results. Traditional voting systems are vulnerable to fraud, errors, and hacking, which can lead to inaccuracies and undermine public trust in the democratic process.

While there have been some attempts to modernize the voting process, including online voting systems and electronic voting machines, these solutions still rely on centralized control and are susceptible to security breaches. Our proposed blockchain-based voting system is a disruptive idea that provides a decentralized, trustless alternative that is transparent, secure, and tamper-proof.

Need for Ethereum Blockchain

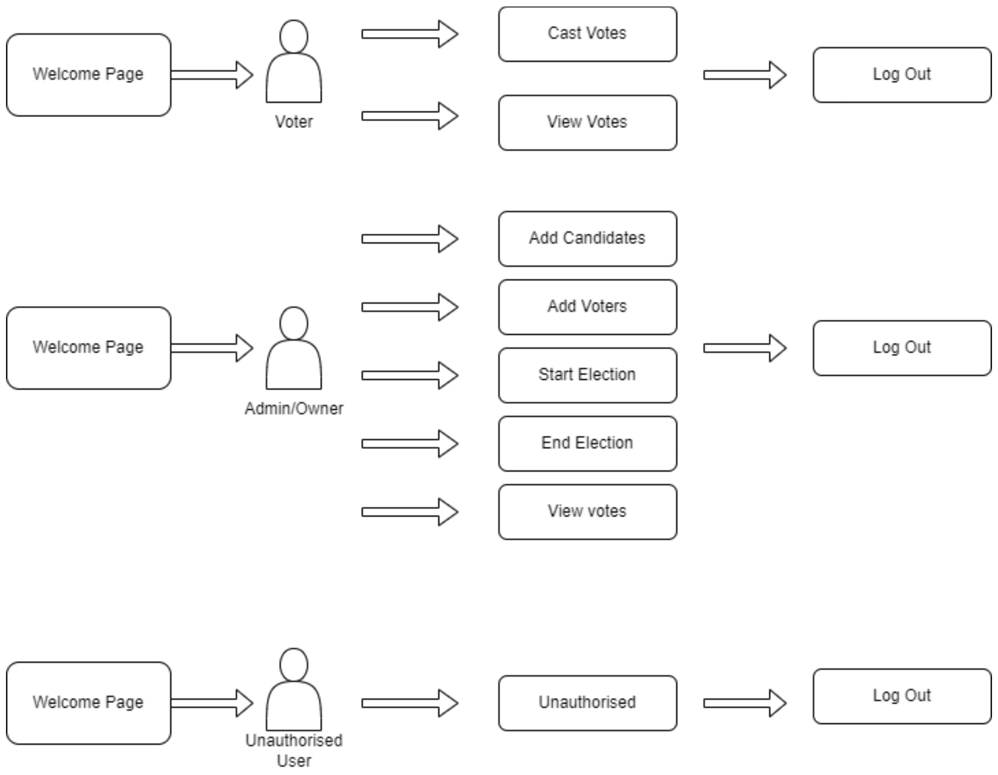
The Ethereum blockchain is a powerful platform for building decentralized applications, and its smart contract functionality makes it the perfect choice for a blockchain-based voting system. Smart contracts are self-executing code that run on the blockchain, and they can be programmed to automate complex processes without the need for intermediaries or centralized control.

In the case of our voting system, smart contracts enable us to create a trustless, decentralized application that is transparent, secure, and tamper-proof. The smart contract code can be programmed to enforce rules and ensure that votes are counted accurately and fairly, without the need for a central authority to oversee the process.

In addition to its smart contract capabilities, Ethereum has a large developer community and a robust infrastructure that make it the best choice for building complex decentralized applications. It also offers several advantages over other blockchain platforms, such as faster transaction processing times and lower transaction fees.

Overall, the Ethereum blockchain provides the ideal foundation for building a secure and transparent voting system that can help address some of the challenges and limitations of traditional voting systems.

Workflow



The Solidity-React based decentralized voting application is designed to be accessible only to two types of users: the owner/admin and voters who have been added by the owner.

The owner will be the address/account on which the voting smart contract is compiled and deployed on the network. Users can access the application using wallets like Metamask.

Upon accessing the application, users are presented with a welcome page that includes a button to enter the voting system. The application checks the current user's address via the connected wallet (Metamask) to determine if the user is an owner, a registered voter, or unauthorized. If the user's address matches the owner's address, the application opens the admin page. From here, the admin can add candidates by inputting their names, add voters by adding their public addresses, and start or end the election. The admin can also see the current status of the election, vote count, and the final result. All of these actions invoke functions in the smart contract in the background.

If the user's address matches one of the registered voters' addresses, the application displays the voting window. However, if the voting has not been started by the owner, the voter will only see a message saying "voting is not started" or "voting ended." If the voting is in progress, the voter can see the list of candidates and cast a vote for one of them. The smart contract prevents users from voting multiple times. After voting is complete, the voter can log out if desired. The voter can view the result only if the owner has ended the election.

In summary, the decentralized voting application restricts access to authorized users only, with the owner having complete control over the voting process. The application ensures transparency and security by leveraging the capabilities of the underlying blockchain technology.

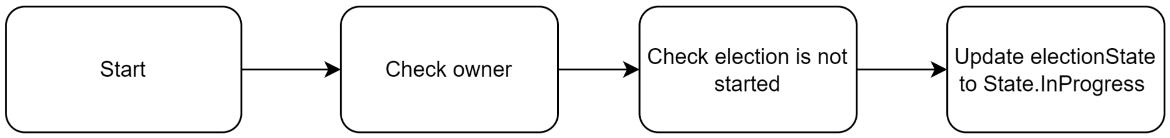
Voting Smart Contract

Election Contract Logic Flow

1. Constructor
   - Initializes owner and electionState variables
   - Calls addCandidate function twice to add two default candidates to the election

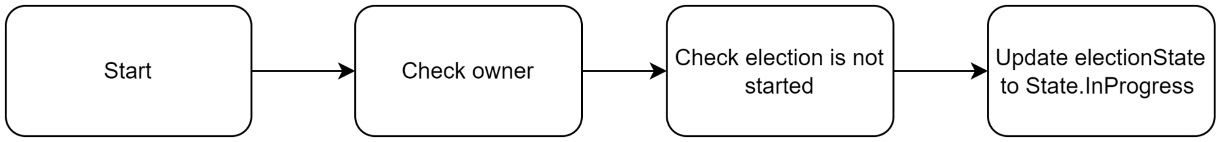
2. startElection function
   - Only the owner of the contract can call this function
   - Can only be called when electionState is NotStarted

- Changes electionState to InProgress


Start
Check owner
Check election is not started
Update electionState to State.InProgress

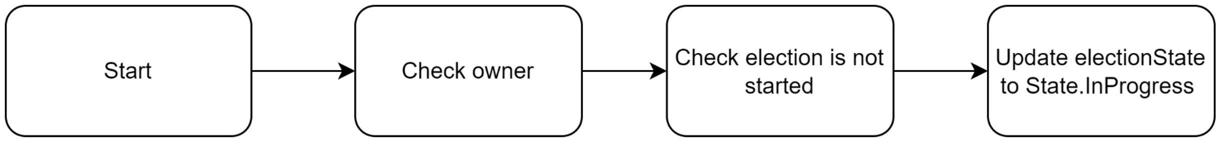
3. endElection function

- Only the owner of the contract can call this function
- Can only be called when electionState is InProgress
- Changes electionState to Ended


Start
Check owner
Check election is not started
Update electionState to State.InProgress

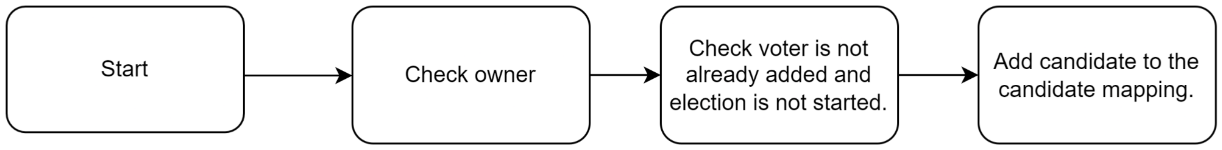
4. addCandidate function

- Only the owner of the contract can call this function
- Can only be called when electionState is NotStarted
- Adds a new candidate to the election by incrementing candidatesCount and adding a Candidate struct to the candidates mapping


Start
Check owner
Check election is not started
Update electionState to State.InProgress

5. addVoter function

- Only the owner of the contract can call this function
- Can only be called when electionState is NotStarted
- Adds a new voter to the election by setting the corresponding address to true in the isVoter mapping

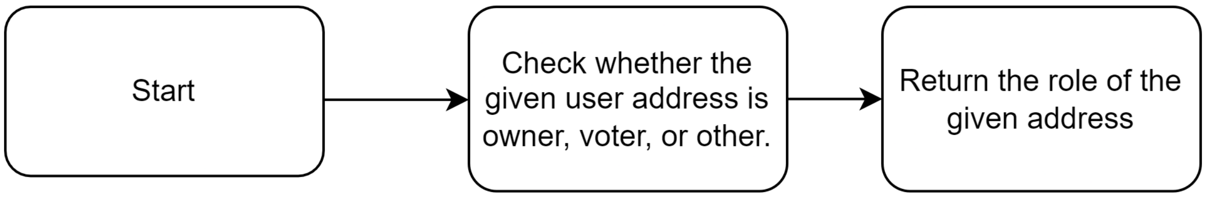

Start
Check owner
Check voter is not already added and election is not started.
Add candidate to the candidate mapping.

6. checkVoter function

- Returns a boolean indicating whether the given address is a registered voter

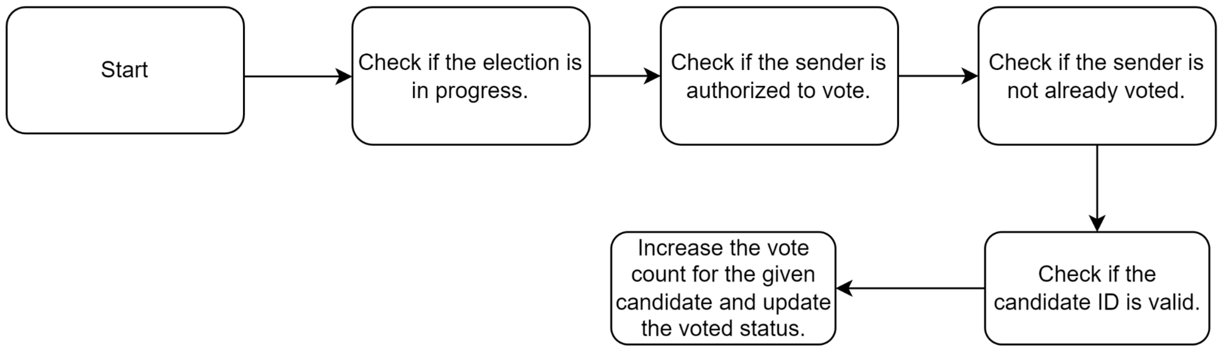
7. getRole function

- Returns an integer representing the role of the given address
  - If the address is the owner, returns 1
  - If the address is a registered voter, returns 2
  - Otherwise, returns 3


Start
Check whether the given user address is owner, voter, or other.
Return the role of the given address

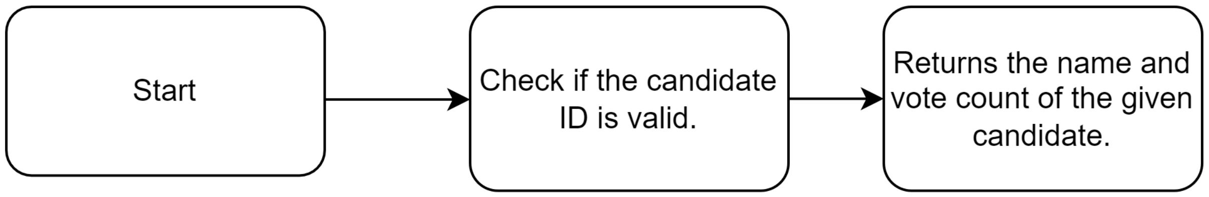
8. vote function

- Can only be called when electionState is InProgress
- Requires that the caller is a registered voter and has not already voted
- Increments the vote count for the selected candidate and marks the caller as having voted
- Emits a Voted event with the candidate ID as the parameter


Start
Check if the election is in progress.
Check if the sender is authorized to vote.
Check if the sender is not already voted.
Check if the candidate ID is valid.
Increase the vote count for the given candidate and update the voted status.

9. getCandidateDetails function

- Returns the name and vote count of the specified candidate
- Requires a valid candidate ID as input


Start
Check if the candidate ID is valid.
Returns the name and vote count of the given candidate.

Smart contract in detail:

Let's go through the different parts of the contract and understand how and when the functions are called and executed.

First, the contract defines an enumeration `State` that has three possible values - `NotStarted`, `InProgress`, and `Ended`. It also defines a `Candidate` struct that has an `id`, `name`, and `voteCount`. Additionally, there is a `Voter` struct that has an `id` and `name`.

The contract has several public variables and mappings. The `owner` is the address of the person who deployed the contract. `electionState` is the current state of the election and is of the `State` enumeration type. `candidates` is a mapping of `Candidate` structs with the `id` as the key. `voted` is a mapping of addresses to booleans to keep track of who has voted. `isVoter` is a mapping of addresses to booleans to keep track of who is authorized to vote. `candidatesCount` and `votersCount` keep track of the number of candidates and voters, respectively.

The constructor initializes the `owner` and sets the `electionState` to `NotStarted`. It also adds two candidates with the names "Candidate 1" and "Candidate 2" using the `addCandidate` function.

The `startElection` function can only be called by the `owner` and changes the `electionState` to `InProgress`. The `endElection` function can also only be called by the `owner` and changes the `electionState` to `Ended`.

The `addCandidate` function can only be called by the `owner` and can only be used when the election is in the `NotStarted` state. It adds a new `Candidate` struct to the `candidates` mapping with the `id` as `candidatesCount` and the provided `_name`. It also increments `candidatesCount`.

The `addVoter` function can only be called by the `owner` and can only be used when the election is in the `NotStarted` state. It adds a new voter to the `isVoter` mapping with the provided address. It checks that the address is not already a voter and increments `votersCount`.

The `checkVoter` function is a public view function that returns `true` if the provided address is a valid voter.

The `getRole` function is a public view function that returns an integer representing the role of the provided address. If the address is the `owner`, it returns 1. If the address is a valid voter, it returns 2. Otherwise, it returns 3.

The `vote` function can only be called when the `electionState` is `InProgress`, the sender is an authorized voter, the sender has not already voted, and the provided `_candidateId` is a valid candidate ID. If all these conditions are met, it increments the `voteCount` of the selected candidate and marks the sender as having voted in the `voted` mapping.

The `getCandidateDetails` function returns the name and vote count of the candidate with the provided `_candidateId`.

Overall, this smart contract provides the basic functionality to conduct an election and ensures that the election is conducted fairly and securely.

Future Enhancements

Our proposed blockchain-based voting system has the potential for several future enhancements to improve its scalability, accessibility, privacy, security, and accuracy. Here are some possible enhancements:

- Integration with other blockchains for scalability and interoperability: Connecting with other blockchain networks can help increase the speed and efficiency of the voting process while also enhancing cross-network compatibility.
- User-friendly interface for ease-of-use and accessibility for voters: A user-friendly interface can encourage more voter participation and engagement, making the voting process more accessible and inclusive.
- Advanced encryption techniques for enhanced privacy and security of voter data: Enhanced encryption techniques can help safeguard voter data and ensure that it remains secure and private.
- Biometric authentication technologies for added security and identity verification: Biometric authentication technologies such as fingerprint or facial recognition can help ensure secure identity verification and prevent fraudulent activity.
- Real-time analytics and reporting for immediate feedback on voting results: Real-time analytics and reporting can provide immediate insight into voting results, increasing transparency and trust in the system.
- Mobile application for increased accessibility and participation for remote voters: A mobile application can help increase accessibility for voters who are unable to vote in person, making the voting process more inclusive and convenient.
- Integration with government databases to automate voter registration and increase participation: Integrating with government databases can help automate the voter registration process and improve participation rates.
- Automatic recounts and audits to ensure accuracy and legitimacy of results: Automatic recounts and audits can help ensure that the voting results are accurate and legitimate, increasing trust in the system.
- Consensus algorithms to improve decentralization and reliability of the system: Consensus algorithms can help ensure the decentralization and reliability of the system, enhancing the transparency and security of the voting process.

Thank You.