WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Decentralized Voting System with React.js Front-End and Blockchain Storage

Saurabh Yadav[1], Harshita Pal[2], Shreya Sahu[3]

[1,2,3]School of Computer Application, Babu Banarasi Das University

ABSTRACT:

This research paper explores the implementation of a decentralized voting system using a front-end application built with React.js, data storage on the blockchain, and connection with Ether.js and Hardhat. A smart contract written in Solidity will be deployed on the blockchain to facilitate the voting process. The paper provides a detailed explanation of the decentralized voting system architecture, the integration of React.js with Ether.js and Hardhat, and the deployment of the smart contract. The system ensures transparency, immutability, and security, while empowering individuals to participate in a decentralized voting process.

Keywords: Hardhat, Meta Mask, Ether.js, Smart Contract,

I. Introduction

In modern democracies, voting is a crucial process for decision-making. However, traditional voting systems often face challenges related to transparency, security, and accessibility. Decentralized voting systems leveraging blockchain technology have emerged as a potential solution to address these challenges. This paper aims to present a decentralized voting system that utilizes a front- end application developed with React.js, data storage on the blockchain, and integration with Ether.js and Hardhat. The system promotes transparency, immutability, and trust in the voting process.

Decentralized Voting System Overview

Definition and Benefits of Decentralization

Decentralization refers to the distribution of power and decision-making authority across a network of participants rather than centralizing it in a single authority or entity. In the context of voting systems, decentralization means removing the reliance on a central authority, such as a government or an election commission, and enabling direct participation and control by the voters themselves.

II. The benefits of decentralization in voting systems are as follows:

a) Transparency: Decentralized voting systems promote transparency by allowing all participants to access and verify the voting process. Since the voting data is recorded on a public blockchain, it becomes immutable and auditable, ensuring the integrity of the system.

b) Security: Decentralized voting systems leverage the security features of blockchain technology to protect against tampering, manipulation, and fraud. The use of cryptographic algorithms ensures the authenticity and immutability of the voting data, enhancing the overall security of the system.

c) Trust: By removing the need for a central authority, decentralized voting systems foster trust among participants. The use of blockchain technology and smart contracts ensures that the voting process is executed exactly as programmed and eliminates the possibility of biased or fraudulent interventions[2].

Accessibility: Decentralized voting systems can improve accessibility by enabling individuals to participate in the voting process remotely, without geographical constraints. This allows for greater inclusivity and convenience for voters who may face difficulties in physically reaching polling stations.

Elimination of Intermediaries: Traditional voting systems often involve multiple intermediaries, such as election officials, administrators, and manual vote counters. Decentralized voting systems eliminate the need for these intermediaries, reducing costs and potential points of failure or manipulation.

Challenges of Traditional Voting Systems

Traditional voting systems face several challenges that decentralized voting systems aim to address:

Lack of Transparency: Traditional voting systems often lack transparency, making it difficult for voters to verify the accuracy and integrity of the voting process. The centralized nature of these systems can lead to concerns about manipulation and fraud.

Vulnerability to Manipulation: Centralized voting systems are susceptible to various forms of manipulation, including tampering with voting machines, altering vote counts, or invalidating ballots. These vulnerabilities undermine the trust and legitimacy of the electoral process.

c) Limited Accessibility: Traditional voting systems may impose limitations on accessibility, such as fixed polling locations, restrictive voting hours, and requirements for physical presence. This can hinder individuals with mobility issues, remote voters, or those residing in remote areas from participating effectively[3].

d) Lack of Privacy: Traditional voting systems often fail to provide adequate privacy to voters. The presence of intermediaries and manual handling of paper ballots can compromise the anonymity of voters, potentially leading to coercion or voter suppression[4].

Introduction to Decentralized Voting Systems

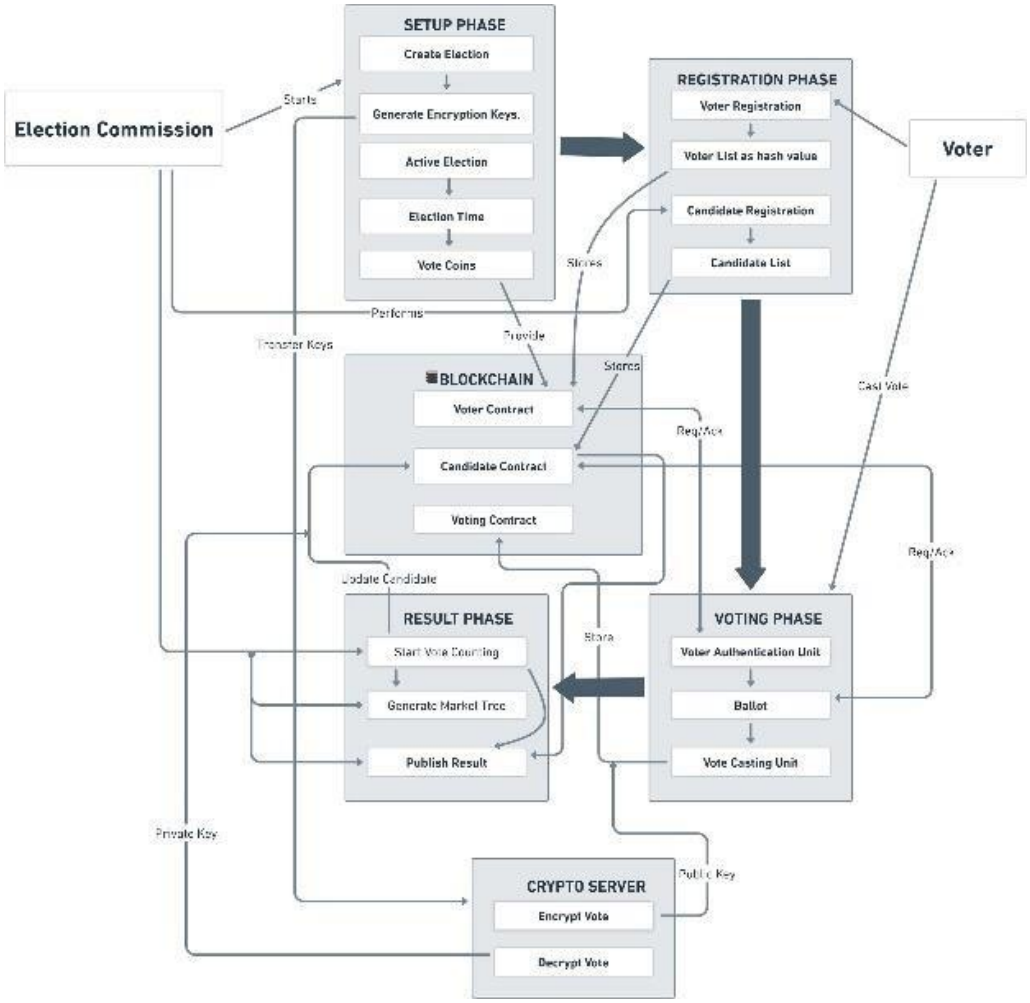

Fig.1 Decentralised Voting Architecture[1]

Decentralized voting systems leverage blockchain technology to create a transparent, secure, and inclusive platform for conducting elections. These systems use the principles of decentralization, cryptography, and consensus algorithms to ensure trust and integrity in the voting process.

In a decentralized voting system, the voting data is recorded on a blockchain, which is a distributed ledger shared among multiple nodes in a network. The blockchain serves as a tamper-resistant and immutable repository of voting records, ensuring transparency and auditability.

Smart contracts, self-executing contracts with predefined rules encoded on the blockchain, play a crucial role in decentralized voting systems. They define the logic and rules of the voting process, including candidate registration, voter authentication, vote casting, and result calculation. Smart contracts eliminate the need for intermediaries and provide a transparent and automated execution of the voting process[5].

Advantages of Decentralized Voting systems

Decentralized voting systems offer several advantages over traditional voting systems:

a) Enhanced Security: By leveraging blockchain technology and cryptographic algorithms, decentralized voting systems provide robust security measures against tampering, manipulation, and fraud. The decentralized nature of the blockchain ensures that no single entity can control or alter the voting data.

b) Immutable and Auditable: Voting data recorded on a blockchain is immutable, meaning it cannot be altered or deleted. This immutability ensures the integrity of the voting process and allows for easy auditing and verification by participants.

c) Trust and Transparency: Decentralized voting systems foster trust among participants by removing the need for a central authority. The transparency of the blockchain allows voters to independently verify the voting process, promoting confidence in the system.

d) Accessibility and Inclusivity: Decentralized voting systems enable individuals to participate in the voting process from anywhere, overcoming geographical barriers. This enhances accessibility and inclusivity, particularly for remote voters, overseas voters, or those with mobility constraints.

e) Efficient and Cost-Effective: By eliminating intermediaries and automating the voting process through smart contracts, decentralized voting systems reduce costs and improve efficiency. The removal of manual processes and paper-based ballots streamlines the entire voting process.

f) Privacy and Anonymity: Decentralized voting systems prioritize the privacy and anonymity of voters. By using cryptographic techniques, these systems ensure that individual votes are encrypted and remain confidential while still contributing to the overall transparency of the process.

In conclusion, decentralized voting systems offer numerous advantages over traditional voting systems, including increased transparency, security, trust, accessibility, and privacy. Leveraging blockchain technology, smart contracts, and decentralized governance, these systems have the potential to revolutionize the democratic process by empowering individuals and ensuring the integrity of elections.

User Interface Considerations and Features

When designing the user interface for a decentralized voting system, the following considerations and features should be taken into account:

a) Candidate Presentation: Display the list of candidates with relevant information, such as their names, pictures, and party affiliations. Consider visual cues to make candidates easily distinguishable.

b) Vote Casting Form: Create a user-friendly form where voters can select their preferred candidate. Provide clear instructions and feedback to guide voters through the process. Implement validation to prevent multiple votes from the same user.

c) Voting Status and Countdown Timer: Show the current voting status, such as whether the voting period is ongoing or closed. Display a countdown timer indicating the remaining time for voting.

d) User Feedback: Provide visual feedback to users after they have cast their votes. This could include a confirmation message, updated vote count for the selected candidate, or a thank you note.

e) Responsive Design: Ensure that the front-end application is responsive and accessible on various devices, such as desktops, tablets, and mobile phones. Responsive design improves the user experience and accessibility for a wider range of voters.

Integration with Ether.js for Blockchain Interaction

To enable interaction with the blockchain in the decentralized voting system, integration with Ether.js is necessary. Ether.js is a JavaScript library that facilitates communication with the Ethereum blockchain. The integration process involves the following steps:

a) Connecting to the Ethereum Network: Use Ether.js to establish a connection to the Ethereum network. This allows the front-end application to communicate with the blockchain and interact with smart contracts[6].

b) Smart Contract Abstraction: Create an abstraction of the deployed Voting smart contract in the front-end application using Ether.js. This abstraction provides an interface to interact with the contract's functions and retrieve data from the blockchain.

c) Interacting with Smart Contract Functions: Utilize Ether.js to call the functions of the smart contract from the front-end application. For example, functions like vote(), getAllVotesOfCandidates(), and getRemainingTime() can be invoked through Ether.js to update the voting data and retrieve relevant information.

d) Handling Transaction Confirmation: Ether.js provides methods to handle transaction confirmation events emitted by the blockchain. These events can be used to update the UI with the latest voting results or provide feedback to users after a successful vote.

Implementation of User Registration and Authentication

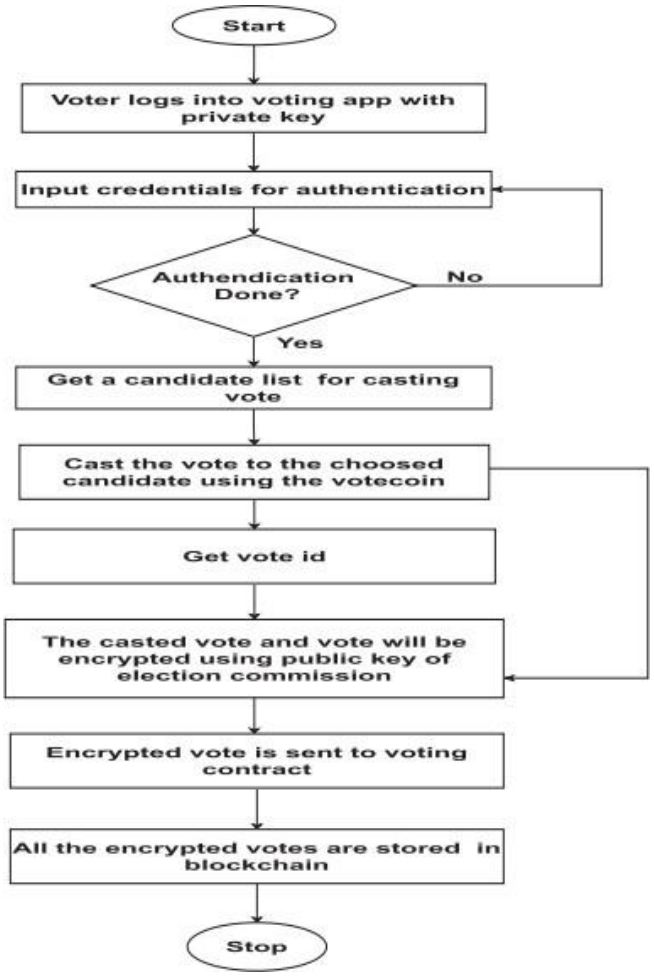

Fig.2 User Authentication in voting[1]

User registration and authentication are essential components of a secure and reliable voting system. Here's how these features can be implemented in the front-end application:

a) User Registration: Create a user registration form where users can provide their necessary details, such as name, email, and identification information.Validate the user inputs and store the registered users' information securely.

b) User Authentication: Implement a login form where users can authenticate themselves with their registered credentials. Use secure authentication mechanisms, such as encrypted passwords or token-based authentication, to ensure user privacy and security.

c) Access Control: Once users are authenticated, implement access control mechanisms to restrict unauthorized access to the voting functionality. For example, only authenticated users should be able to access the vote casting form and view the voting results.

d) Session Management: Manage user sessions to maintain the authentication state throughout the user's interaction with the application. This allows users to stay logged in and securely access the voting features without repeated authentication.

e) Security Considerations: Ensure that user registration and authentication processes follow best practices for security, such as using strong encryption for passwords, implementing measures against brute-force attacks, and protecting sensitive user data.

By implementing user registration and authentication in the front-end application, the decentralized voting system can ensure that only eligible and authenticated users can participate in the voting process, enhancing security and integrity.

In conclusion, front-end development with React.js plays a crucial role in creating a user- friendly and intuitive interface for the decentralized voting system. It involves designing and developing UI components, considering user interface aspects, integrating with Ether.js for blockchain interaction, and implementing user registration and authentication features to ensure security and accessibility.

Front-End Development with React.js

Introduction to React.js

React.js is a popular JavaScript library for building user interfaces. It allows developers to create reusable UI components and efficiently manage the application's state. React.js follows a component-based architecture, where each component encapsulates its own logic and rendering.

3 User Interface Considerations and Features

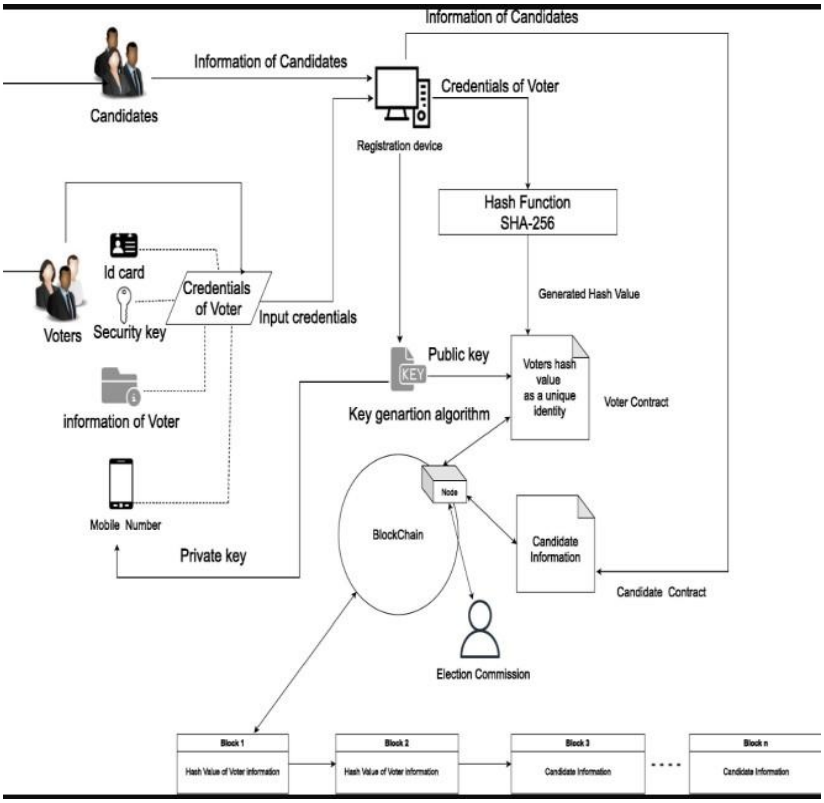

Fig [3] User Interaction with blockchain network

When designing the user interface for a decentralized voting system, the following considerations and features should be taken into account:

a) Candidate Presentation: Display the list of candidates with relevant information, such as their names, pictures, and party affiliations. Consider visual cues to make candidates easily distinguishable.

b) Vote Casting Form: Create a user-friendly form where voters can select their preferred candidate. Provide clear instructions and feedback to guide voters through the process. Implement validation to prevent multiple votes from the same usercVoting Status and Countdown Timer: Show the current voting status, such as whether the voting period is ongoing or closed. Display a countdown timer indicating the remaining time for voting.

d) User Feedback: Provide visual feedback to users after they have cast their votes. This could include a confirmation message, updated vote count for the selected candidate, or a thank you note.

e) Responsive Design: Ensure that the front-end application is responsive and accessible on various devices, such as desktops, tablets, and mobile phones. Responsive design improves the user experience and accessibility for a wider range of voters.

Integration with Ether.js for Blockchain Interaction

To enable interaction with the blockchain in the decentralized voting system, integration with Ether.js is necessary. Ether.js is a JavaScript library that facilitates communication with the Ethereum blockchain. The integration process involves the following steps:

a) Connecting to the Ethereum Network: Use Ether.js to establish a connection to the Ethereum network. This allows the front-end application to communicate with the blockchain and interact with smart contracts.

b) Smart Contract Abstraction: Create an abstraction of the deployed Voting smart contract in the front-end application using Ether.js. This abstraction provides an interface to interact with the contract's functions and retrieve data from the blockchain.

c) Interacting with Smart Contract Functions: Utilize Ether.js to call the functions of the smart contract from the front-end application. For example, functions like vote(), get AllVotesOfCandidates(),and.getRemainingTi me() can be invoked through Ether.js to update the voting data and retrieve relevant information.

d) Handling Transaction Confirmation: Ether.js provides methods to handle transaction confirmation events emitted by the blockchain. These events can be used to update the UI with the latest voting results or provide feedback to users after a successful vote.

III. Implementation of User Registration and Authentication

User registration and authentication are essential components of a secure and reliable voting system. Here's how these features can be implemented in the front-end application:

a) User Registration: Create a user registration form where users can provide their necessary details, such as name, email, and identification information. Validate the user inputs and store the registered users' information securely.

b) User Authentication: Implement a login form where users can authenticate themselves with their registered credentials. Use secure authentication mechanisms, such as encrypted passwords or token-based authentication, to ensure user privacy and security.

c) Access Control: Once users are authenticated, implement access control mechanisms to restrict unauthorized access to the voting functionality. For example, only authenticated users should be able to access the vote casting form and view the voting results.

d) Session Management: Manage user sessions to maintain the authentication state throughout the user's interaction with the application. This allows users to stay logged in and securely access the voting features without repeated authentication.

e) Security Considerations: Ensure that user registration and authentication processes follow best practices for security, such as using strong encryption for passwords, implementing measures against brute-force attacks, and protecting sensitive user data.

By implementing user registration and authentication in the front-end application, the decentralized voting system can ensure that only eligible and authenticated users can participate in the voting process, enhancing security and integrity.

IV. Overview of Blockchain Technology

Blockchain technology is a distributed and decentralized ledger that records transactions across multiple computers or nodes in a network. It provides a secure and transparent way to store and verify data without the need for a central authority. Each transaction is added to a block, which is then cryptographically linked to the previous block, forming a chain of blocks (hence the name blockchain).

Advantages of Using Blockchain for Voting Systems

Using blockchain technology in voting systems offers several advantages:

a) Transparency: The transparent nature of blockchain allows all participants to view and verify the transactions recorded on the blockchain. This transparency promotes trust and ensures that the voting process is fair and free from manipulation.

b) Security: Blockchain employs cryptographic techniques to ensure the security and integrity of data. Each block is cryptographically linked to the previous block, making it extremely difficult to alter or tamper with the stored information. This tamper-resistant nature enhances the security of the voting system.

c) Immutability: Once data is stored on the blockchain, it becomes immutable and cannot be changed or deleted. This immutability provides a reliable and auditable record of votes, eliminating the possibility of fraudulent activities or data manipulation[8].

d) Decentralization: By removing the need for a central authority, blockchain-based voting systems empower individual voters and reduce the reliance on intermediaries. Decentralization increases the resilience of the system and ensures that no single entity can control or manipulate the voting process.

e) Accessibility: Blockchain-based voting systems can be accessed from anywhere, enabling remote voting and increasing accessibility for voters who face physical limitations or reside in remote locations. This promotes inclusivity and improves voter turnout.

Introduction to Ether.js and Its Role in Interacting with Ethereum Blockchain

Ether.js is a JavaScript library that simplifies the interaction with the Ethereum blockchain. It provides a convenient way to connect to the Ethereum network, interact with smart contracts, and send transactions. Ether.js abstracts the complexities of the Ethereum protocol, making it easier for developers to build applications that integrate with the blockchain.Ether.js facilitates the following interactions with the Ethereum blockchain:

a) Connecting to the Ethereum Network: Ether.js enables the front-end application to connect to an Ethereum node, either by using a local node or a hosted service provider. This connection allows the application to interact with the blockchain.

b) Smart Contract Interaction: Ether.js provides a convenient way to interact with smart contracts deployed on the Ethereum blockchain. It allows developers to call functions, retrieve data, and listen to events emitted by the smart contracts.

c) Transaction Handling: Ether.js simplifies the process of sending transactions to the Ethereum blockchain. It handles the creation and signing of transactions, estimates gas fees, and manages the transaction lifecycle.

d) Event Listening: Ether.js allows the front- end application to listen for events emitted by smart contracts. These events can be used to update the user interface with real-time information, such as vote counts or changes in the voting status.

V. Implementation of Data Storage on the Blockchain

In the decentralized voting system, data storage is a critical aspect that ensures the transparency and integrity of the voting process. The Voting smart contract provided earlier in the research paper demonstrates the implementation of data storage on the blockchain

The smart contract stores voting data in the form of candidate names and their respective vote counts. Each candidate is represented by a struct, which includes a string variable for the name and a uint256 variable for the vote count. The candidates' data is stored in an array.

When a user casts a vote, the smart contract updates the vote count for the selected candidate. The voting data is permanently stored on the blockchain and can be accessed and verified by anyone.

VI. Securing Data Integrity and Immutability

The decentralized nature of the blockchain ensures the security, integrity, and immutability of the voting data. However, additional measures can be implemented to enhance data security:

a) Access Control: Implement access control mechanisms in the smart contract to restrict who can update the voting data. For example, only authorized addresses or the contract owner can add candidates or update vote counts.

b) Secure Smart Contract Development: Follow secure coding practices when developing the smart contract to minimize vulnerabilities. Use the latest version of the Solidity programming language and follow best practices for contract design, such as avoiding the use of deprecated functions and libraries.

c) Code Auditing: Conduct a thorough code review or engage external auditors to identify potential security vulnerabilities or loopholes in the smart contract. This helps ensure that the contract is secure and resilient against attacks.

d) External Oracle Integration: In some cases, external data sources may be required to verify certain information, such as voter eligibility or identity. Integration with trusted external oracles can provide additional layers of security and validation.

e) Testing and Simulation: Perform comprehensive testing and simulation of the decentralized voting system to identify any potential weaknesses or vulnerabilities. This includes testing various scenarios, stress testing the system, and validating its behavior under different conditions.

By implementing these measures, the decentralized voting system can ensure the security, integrity, and immutability of the voting data stored on the blockchain, fostering trust among participants and reinforcing the reliability of decentralised the system.

Smart Contract Deployment with Hardhat

VII. Introduction to Smart Contracts

Smart contracts are self-executing contracts with the terms of the agreement directly written into code. They are deployed on a blockchain network and automatically execute predefined actions when certain conditions are met. Smart contracts eliminate the need for intermediaries and provide transparency, security, and immutability to the agreements they represent.

Solidity Programming Language Overview

Solidity is a high-level programming language specifically designed for writing smart contracts on the Ethereum blockchain. It has a syntax similar to JavaScript and supports features like contract inheritance, libraries, and complex data structures.

VIII. Key features of Solidity include:

a. Contract Definition: Solidity allows developers to define contracts, which act as the building blocks for smart contracts. Contracts encapsulate data and functions related to a specific agreement or application.

b. Data Types: Solidity supports various data types, including integers, booleans, strings, addresses, and arrays. It also allows for the creation of custom data structures using structs and enums.

c. Functions and Modifiers: Solidity allows the definition of functions that can modify the contract's state or retrieve data from it. Modifiers can be used to enforce access controls or perform common operations before executing a function.

d. Events and Logging: Solidity provides an event system to emit events from smart contracts. Events can be used to notify external systems or trigger actions in the front-end application.

Writing and Deploying the Voting Smart Contract

To deploy the Voting smart contract using Hardhat, follow these steps:

a) Set Up Hardhat: Install Hardhat and create a new project directory. Set up the necessary configuration files, such as hardhat.config.js and solc.config.js, to define the network settings and Solidity compiler options.

b) Write the Smart Contract: In a Solidity file (e.g., Voting.sol), define the Voting contract using the provided smart contract code. Make sure to import any necessary Solidity libraries or dependencies.

c) Compile the Contract: Use Hardhat's compile command to compile the smart contract code. This generates the necessary bytecode and ABI (Application Binary Interface) artifacts.

d) Deploy the Contract: Define a deployment script (e.g., deploy.js) to deploy the compiled contract to a specific Ethereum network. The deployment script should include the necessary parameters, such as the candidate names and the duration of the voting period.

e) Run the Deployment: Execute the deployment script using Hardhat's deploy command. This will deploy the smart contract to the specified Ethereum network.

Integration of the Smart Contract with React.js and Ether.js

To integrate the deployed smart contract with the front-end application built with React.js and Ether.js, follow these steps:

a) Import the Smart Contract ABI: In the front- end application, import the ABI artifact generated during the smart contract compilation. The ABI provides an interface for interacting with the smart contract's functions and events.

b) Connect to the Ethereum Network: Use Ether.js to establish a connection to the Ethereum network. This connection allows the front-end application to interact with the deployed smart contract.

c) Create Contract Instances: Using the imported ABI and the contract's address, create instances of the smart contract in the front-end application. These instances can be used to call the contract's functions and listen for events.

d) Invoke Smart Contract Functions: Utilize Ether.js to invoke the functions of the deployed smart contract from the front-end application. For example, you can call the vote() function to cast a vote or getAllVotesOfCandidates() to retrieve the vote counts of all candidates.

e) Update User Interface: Based on the responses received from the smart contract function calls, update the user interface to reflect the current voting status, display the vote counts, and provide real-time feedback to users.

IX. Ensuring Transparency and Auditability through the Smart Contract

The Voting smart contract deployed on the blockchain provides transparency and auditability to the decentralized voting system. The following features ensure the integrity of the voting process:

a) Immutable Storage: As discussed earlier, the smart contract stores voting data on the blockchain, ensuring its immutability. Once a vote is recorded, it cannot be altered or deleted, providing a transparent and auditable record of the voting process.

b) Public Visibility of Contract and Functions: The smart contract and its functions are publicly visible on the blockchain. This enables anyone to inspect the contract's code and verify its behavior. Transparency and openness foster trust among participants in the voting system.

c) Event Emission: The smart contract emits events for important actions or state changes. These events can be captured by the front-end application using Ether.js and used to provide real-time updates to the user interface. Events also contribute to transparency and auditability by documenting key activities on the blockchain.

d) External Verification: Interested parties can verify the accuracy of the voting process by independently interacting with the smart contract. They can retrieve voting data, check the vote counts, and compare the results with the publicly available information.

e) Accessibility and Participation: The decentralized nature of the voting system allows eligible participants to access and engage in the voting process, ensuring inclusivity and democratic participation. The transparency provided by the smart contract builds trust among voters and encourages their active involvement.

X. Evaluation and Security Considerations :-

Security Measures in the Decentralized Voting System

The decentralized voting system implemented using blockchain technology and smart contracts incorporates several security measures to protect the integrity and confidentiality of the voting process:

a) Cryptographic Security: Blockchain utilizes cryptographic algorithms to secure transactions and data. This ensures that votes and other sensitive information are encrypted and protected from unauthorized access[10].

b) Access Control: The smart contract can include access control mechanisms to restrict certain functions or data to authorized entities only. This prevents unauthorized manipulation or tampering of the voting system.

c) Immutable and Transparent Record: The immutability of the blockchain ensures that voting data cannot be altered or deleted once recorded. The transparent nature of the blockchain allows participants to verify the integrity of the voting process.

d) Voter Authentication: The implementation of user registration and authentication in the front- end application ensures that only eligible voters can cast their votes. This helps prevent fraudulent voting activities.

e) Event Logging and Auditability: By emitting events in the smart contract, the voting system provides a clear audit trail of important actions and state changes. This allows for transparent auditing and verification of the voting data.

Verification and Auditing of Voting Data on the Blockchain

One of the key advantages of a decentralized voting system is the ability to verify and audit voting data stored on the blockchain. The following mechanisms enable verification and auditing:

a) Public Accessibility: The blockchain is a public and distributed ledger, allowing anyone to access and view the stored data. This promotes transparency and enables independent verification of the voting process.

b) Immutable Data: Once recorded on the blockchain, voting data is immutable and cannot be altered or tampered with. This ensures the integrity and reliability of the data during the auditing process.

c) Smart Contract Events: The smart contract emits events for important actions and state changes. These events serve as an audit trail, documenting the key activities in the voting system. Auditors can analyze these events to verify the consistency and accuracy of the voting process.

d) Comparison with External Data Sources: The voting data stored on the blockchain can be compared and cross-referenced with external data sources to ensure consistency. For example, voter registration data can be compared with official voter registration records to verify voter eligibility.

XI. Potential Vulnerabilities and Mitigations

While decentralized voting systems offer enhanced security compared to traditional voting systems, they can still be susceptible to certain vulnerabilities. Here are some potential vulnerabilities and their corresponding mitigations:

a) Sybil Attacks: Sybil attacks involve creating multiple fake identities to manipulate the voting process. To mitigate this, the voting system can implement identity verification measures, such as KYC (Know Your Customer), to ensure that each voter has a unique and verified identity[9].

b) Insider Attacks: Insider attacks occur when individuals with privileged access manipulate the voting system. Implementing strict access controls and regular audits can help mitigate this risk.

c) Malicious Smart Contracts: Smart contracts can contain vulnerabilities that could be exploited by attackers. Conducting rigorous code reviews, utilizing security best practices, and performing third-party audits can help identify and mitigate these vulnerabilities.

d) Network Attacks: The underlying blockchain network can be susceptible to various network attacks, such as DDoS (Distributed Denial of Service) attacks or 51% attacks. Deploying the voting system on a reputable and secure blockchain network, such as Ethereum, can mitigate these risks[11].

e) Front-End Security: The front-end application used for user interaction can have vulnerabilities that may compromise the security of the voting system. Implementing secure coding practices, regular security testing, and staying up-to-date with security patches can help mitigate these risks.

References :

[1] DVTChain: A blockchain-based decentralized mechanism to ensure the security of digital voting system voting system Syada Tasmia Alvi a,⇑ , Mohammed Nasir Uddin b , Linta Islam b , Sajib Ahamed b aDepartment of CSE, Daffodil International University, Bangladesh

[2] Blockchain for Electronic Voting System—Review andOpenChallenges:https://www.ncbi.nlm.nih.gov/pmc/a rticles/PMC8434614/

[3] Alvarez, R. M., & Hall, T. E. (2008). Voting without borders: The impact of electronic voting on turnout. Electoral Studies, 27(4), 578-591.

[4] Banerjee, S., & Sundararajan, A. (2018). The privacy of voting: A survey. ACM Computing Surveys, 51(2), 1- 34.

[5] Gastinger, M., Lafourcade, P., & Tazi-Lkhnini, Y. (2018). Smart contracts for decentralized voting systems. arXiv preprint arXiv:1801.04404.

[6] https://docs.ethers.org/v5/

[7] Juels, A., & Rivest, R. L. (2007). The sybil attack. In Security in computer networks (pp. 174-187). Springer.

[8] Singh, H. K., Kumar, S., & Singh, J. (2021). Blockchain for electronic voting system—Review and open challenges. PeerJ Computer Science, 7, e840.

[9] Rivest, R. L., & Wack, J. P. (2006). How to build a sybil-resistant distributed voting system. In International conference on financial cryptography and data security (pp. 1-12). Springer.

[10] Juels, A., & Rivest, R. L. (2007). The sybil attack. In Security in computer networks (pp. 174-187). Springer.

[11] Meiklejohn, S., et al. (2013). A Sybil-resistant voter registration system. In USENIX Security Symposium (pp. 301-316).