|  |  |  |
| --- | --- | --- |
| NO | NAMES | REG NO |
| 1 | NIYIMURERA Janvier | 23RP00815 |
| 2 | NIYIRANSHUTI Elizabeth | 23RP01198 |
| 3 | DUSINGIZEYEZU Jeannete | 23RP01135 |
| 4 | NYIRANSANZUMUHIRE Chantal | 23RP01671 |

**SMART CONTRACT NAME: smart contract for managing secure and transparent voting processes**

**INTRODUCTION**

The rise of blockchain technology has brought significant advancements in creating secure and transparent digital systems. One such application is in the realm of voting, where the integrity and transparency of the process are paramount. Traditional voting systems are often plagued by issues such as voter fraud, tampering, and lack of transparency, which can undermine the democratic process. By leveraging the immutable and decentralized nature of blockchain, a more secure and trustworthy voting system can be developed.

The proposed smart contract for managing secure and transparent voting processes aims to address these issues by providing a platform where votes are securely cast, transparently counted, and permanently recorded on the blockchain. This application ensures that only authorized voters can participate, and once a vote is cast, it is irrevocable and visible to all participants. The transparency provided by the blockchain allows for real-time auditing and verification of the election process, thereby enhancing trust among voters and stakeholders.

Moreover, this smart contract for managing secure and transparent voting processes is designed to be user-friendly and efficient. The system facilitates the easy addition of candidates, authorization of voters, and casting of votes, ensuring a smooth and accessible voting experience. Additionally, the use of smart contracts automates the vote counting process and the publication of results, reducing the potential for human error and manipulation. This innovative approach not only modernizes the voting process but also upholds the principles of democracy by ensuring every vote is counted accurately and transparently.

**System Overview**

The smart contract-based voting application includes the following key features and components:

1. **Functional requirements:**

**User Management:**

**Candidate Management:**

* Adding new candidates by the contract owner.
* Storing candidate details on the blockchain.
* Creation of user accounts.
* Authorization of voters by the contract owner.

**Voting Process:**

* Authorized voters can cast their votes.
* Each voter can only vote once.

**Vote Counting and Results:**

* Real-time publication of voting results.
* Transparent display of candidate vote counts.

1. **Security and Transparency:**

* Immutable records of votes stored on the blockchain.
* Only authorized voters can participate in the voting process.
* Events emitted for every vote cast to ensure transparency.

1. **Non-Functional Requirements:**

* High security to prevent unauthorized access and tampering.
* Scalability to handle a large number of voters and candidates.
* Minimal transaction fees for interacting with the smart contract.

1. **Component Details:**
2. **Blockchain Platform:**

* Selection of a blockchain that supports smart contracts.
* Deployment of the voting smart contract on the chosen platform.

1. **Smart Contract:**

* Development and deployment of the core voting logic using Solidity.
* Functions for adding candidates, authorizing voters and casting votes.

1. **User Interface (UI):**

* Development of a responsive web application for user interactions.
* Features for candidate addition, voter authorization, and real-time results display.

1. **Hosting and Deployment:**

* Hosting the web application on reliable platforms.
* Optional decentralized hosting using IPFS for enhanced security and availability.

1. **Backend Services:**

* API development for interacting with the smart contract.
* Optional backend logic for additional functionality (e.g., user management).

**Data Flow**

* + **User** creates an account and logs in to the application.
  + **Admin** adds candidates to the election by calling the addCandidate function.
  + **Authorized Voter** casts a vote by calling the vote function.
  + The voter's vote is recorded in the voters mapping.
  + **Smart Contract** automatically counts votes as they are cast.

1. **Security and Verification:**
   * **Smart Contract** verifies if the user is authorized and hasn't voted before.
   * Data: User's Ethereum address
   * Flow: Verification logic ensures data integrity and prevents double voting.
2. **Results Announcement:**
   * **Admin** announces the election results.
   * Data: Candidate IDs and their respective vote counts
   * Flow: Results are fetched from the smart contract and displayed on the UI.
     + 1. Integration Requirements
3. **Backend Services Integration:**

* **API Development:** Create backend services using Node.js/Express to handle additional business logic and API endpoints.

1. **Wallet Integration:**

* **MetaMask Integration:** Integrate MetaMask to enable users to securely interact with the smart contract using their Ethereum wallets.

1. **Documentation and User Support:**

* **Developer Documentation:** Provide comprehensive documentation for developers to understand and extend the smart contract and application.
* **User Guide:** Create a user guide to help end-users understand how to interact with the application.

**Success Criteria**

1. **Secure and Transparent Voting Process:**
   * The application must ensure that all votes are securely cast, transparently counted, and immutably recorded on the blockchain.
2. **User-Friendly Experience:**
   * The application must provide an intuitive and user-friendly interface for all users, including voters and administrators.
3. References

<https://docs.soliditylang.org/en/v0.8.26/>

<https://www.simplilearn.com/tutorials/blockchain-tutorial/what-is-solidity-programming>

<https://docs.arbitrum.io/build-decentralized-apps/quickstart-solidity-hardhat>