DECENTRALIZED VOTING SYSTEM

A PRATICAL BASED REPORT ON BCT (BCS613A)

Submitted to

Visvesvaraya Technological University

BELAGAVI - 590 018

by

Prathishta

4SU22AD038

Under the guidance of

Mr. Amith K S

Assistant Professor

in partial fulfillment of the requirements for the award of the degree of

Bachelor of Engineering

Department of Artificial Intelligence & Data Science

SDM INSTITUTE OF TECHNOLOGY

UJIRE - 574 240

2024-2025

SDM INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belagavi)
UJIRE – 574 240

Department of Artificial Intelligence & Data Science

CERTIFICATE

It is certified that the project titled **"Decentralized Voting System** " conducted by **Ms. Prathishta (USN: 4SU22AD038)**, a Bonafide student of **SDM Institute of Technology**, Ujire, has been carried out in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in **Artificial Intelligence & Data Science** from Visvesvaraya Technological University, Belagavi, during the academic year 2024-2025. The report has been approved as it satisfies the academic requirements prescribed for the practical based learning for the Blockchain Technology (BCS613A).

Mr. Amith K S	**Mrs. Veena Bhat**
Assistant Professor	Head of the Department

Acknowledgement

It is my pleasure to express our heartfelt thanks to **Mr. Amith K S** Assistant Professor, Department of Artificial Intelligence and Data Science, for his supervision and guidance which enabled us to understand and develop this project.

I am indebted to **Dr. Ashok Kumar T**, Principal, and **Mrs. Veena Bhat**, Head of the Department, for their advice and suggestions at various stages of the work. I also extend our heartfelt gratitude to Management of SDM Institute of Technology, Ujire, for providing me with a good learning environment, library and laboratory facilities. I appreciate the help and the support rendered by the teaching and non- teaching staff of Artificial Intelligence and Data Science and Engineering.

Lastly, I take this opportunity to offer our regards to all of those who have supported me directly or indirectly in the successful completion of this project work.

Prathishta

ABSTRACT

The Decentralized Voting System is a blockchain-based solution designed to enhance the security, transparency, and reliability of elections. Traditional voting systems often face challenges such as tampering, lack of public trust, centralized control, and limited verifiability. This project addresses these issues by using the Ethereum blockchain and Solidity smart contracts to create a tamper-proof, decentralized voting mechanism. The system is developed entirely using Solidity within the Remix IDE environment, enabling a simple and focused approach to smart contract-based election management. It allows an administrator to register candidates, initiate and end the voting phase, and retrieve final results. Each voter is permitted to cast one vote, and all actions are securely recorded on the blockchain to prevent duplication and ensure transparency. The smart contract also features built-in logic to detect a draw situation in the event of tied votes. Test cases were conducted within Remix IDE to validate functionality such as vote casting, restriction of double voting, candidate registration timing, and correct winner calculation. This prototype demonstrates the practical potential of decentralized voting systems using minimal infrastructure. While effective at a functional level, future enhancements may involve adding voter authentication, improving privacy, and integrating real-world identity verification mechanisms for broader applicability.

Tables of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This project presents a blockchain-based voting system developed using Solidity in Remix IDE, aiming to ensure secure, transparent, and tamper-proof elections. It allows an admin to manage candidates and voting phases, while each voter can cast only one verifiable vote. Results are computed automatically, and all data is immutably recorded on the Ethereum blockchain, ensuring fairness without centralized control.

1.1 Overview of the Project

The Decentralized Voting System is a blockchain-based solution built with Solidity and Remix IDE to ensure secure, transparent, and tamper-proof elections. It features admin-controlled candidate registration, voting period management, one-person-one-vote enforcement, and automatic winner calculation with tie detection. By recording votes immutably on the Ethereum blockchain, it addresses key flaws in traditional systems and enhances electoral integrity.

1.2 Problem Statement

Modern voting systems often face fraud, tampering, and lack of transparency due to centralized control. These issues erode trust, especially in regions with weak governance. The Decentralized Voting System addresses these problems by using blockchain to eliminate central authority, ensure real-time verifiability, and securely record each vote immutably and transparently.

1.3 Objectives of the Project

The primary goals of the Decentralized Voting System project are:

1. Develop a secure and tamper-resistant voting platform using Ethereum blockchain and Solidity smart contracts that ensures only eligible users can vote once by tracking blockchain addresses.
2. Enable administrators to manage election phases, including candidate registration and controlling the voting period.
3. Provide real-time vote counting with automatic winner determination and draw detection, while eliminating common issues like voter fraud, administrative inefficiencies, and lack of transparency.

1.4 Scope of the Project

This project involves developing a prototype decentralized voting system. The current implementation includes smart contracts for vote registration, candidate management, and winner determination. Administrative controls allow managing the voting process and candidate registration. While the system currently does not support biometric voter authentication, large-scale election scalability, or integration with official electoral authorities, these aspects are planned as potential future improvements.

1.5 Methodology

The project methodology involved studying Ethereum blockchain fundamentals and smart contract security to design a secure voting system.

- Solidity was used to develop smart contracts for voter registration, candidate management, vote casting, and automatic tallying, ensuring immutability and transparency.
- Development and deployment were carried out entirely within Remix IDE, allowing efficient coding, testing, and debugging.
- The approach prioritized best security practices to protect against common vulnerabilities, resulting in a simple yet robust decentralized voting platform.
- The primary research involved studying Ethereum's decentralized architecture, smart contract principles, and Solidity programming. Focus was placed on understanding the Ethereum Virtual Machine (EVM), gas fees, and best practices for secure and efficient smart contract development.
- The research also covered Solidity's syntax, control structures, and security considerations. Remix IDE was explored as the main environment for writing, compiling, deploying, and debugging smart contracts.
- The development focused on writing the voting system's smart contracts in Solidity, defining voter and administrator roles, and implementing key functions such as voter registration, candidate management, voting, and vote tallying.
- Deployment and testing were conducted entirely through Remix IDE, which simplified the workflow and allowed direct compilation and deployment of smart contracts. This approach prioritized core smart contract functionality, enabling faster development and easier maintenance without relying on additional tools.

Chapter 2

Literature Survey

A literature review is a summary and critical analysis of existing research and publications related to a specific topic or project. It helps identify gaps, trends, and key findings in the field. In academic projects, it provides a foundation by comparing past work with the current problem being addressed. It also justifies the relevance and uniqueness of the proposed solution.

2.1 Introduction to Literature Survey

This literature survey explores how blockchain can enable secure, transparent, and tamper-proof voting. It reviews key research and pilot projects, while also addressing the technical, legal, and infrastructural challenges that hinder widespread adoption.

2.2 Existing Systems or Technologies

Noizat [1] explored the design of secure and verifiable voting systems using blockchain. The paper discussed using public ledgers to create an auditable vote trail, while proposing anonymization techniques to preserve voter privacy.

Ayed, et al. [2] did a comprehensive survey that analysed various blockchain voting models and compared their resistance to fraud, coercion, and double voting. It also outlined key challenges like user authentication and scalability.

McCorry, et al. [3] proposed an Ethereum-based voting protocol that ensures privacy, verifiability, and decentralization. It introduced zero-knowledge proofs and ring signatures for voter anonymity.

Hjalmarsson et al. [4] study implemented a voting prototype using Ethereum smart contracts. It emphasized the usability of public blockchain platforms and discussed trade-offs between transparency and voter anonymity.

Al-Khateeb et al. [5] developed and tested a blockchain voting DApp using Solidity, web3.js, and Ethereum. Their prototype was tested on Ropsten testnet and included vote encryption and event monitoring.

2.3 Gaps in Existing Solution

Despite the promising nature of blockchain voting systems, key challenges persist:

1. **Scalability**: Public blockchains like Ethereum face slow transaction speeds and high gas fees.
2. **Voter Privacy**: Balancing transparency with anonymity remains technically complex.
3. **Authentication**: Most systems lack robust identity verification beyond crypto wallets.
4. **Real-World Adoption**: Few large-scale deployments exist, limiting practical evaluation.

2.4 Summary of Literature Survey

The Decentralized Voting System uses Ethereum smart contracts to enforce one-person-one-vote, record votes immutably, and automate result calculation. Its modular design separates admin and user roles, ensuring security and flexibility for future enhancements like biometric login or mobile support.

Chapter 3

Software and Hardware Requirements

The successful implementation of a decentralized voting system relies on an integrated suite of software tools and compatible hardware. This chapter details the technical environment required to develop, test, and deploy the voting system. These requirements ensure that the smart contract, blockchain interactions, and frontend interface run smoothly and securely, even in a simulated environment.

3.1 Software Requirements

The software components required for the Decentralized Voting Systems are listed in Table 3.1

Table 3.1 Software requirements

Sl. No.	Category	Description
01	Operating System	Windows 10 / Linux / macOS
02	Smart Contract Language	Solidity
03	Development Environment	Remix IDE

3.2 Hardware Requirements

The hardware setup is critical to ensure seamless functionality of the Decentralized Voting System are referred in Table 3.2.

Table 3.2 Hardware requirements

Sl. No.	Category	Description
01	Processor	Intel i5 or equivalent (64-bit)
02	RAM	8 GB
03	Storage	Minimum 10 GB free space
04	Display	1280 × 720 resolution or higher
05	System Architecture	64-bit system

Chapter 4

System Design

This chapter outlines the architecture and design of the Decentralized Voting System, detailing component interactions, module functions, and data flow, supported by diagrams to illustrate system behaviour and user interactions.

4.1 Architecture Design

The Decentralized Voting System adopts a three-layer decentralized architecture that integrates blockchain backend logic, wallet authentication, and a browser-based frontend.

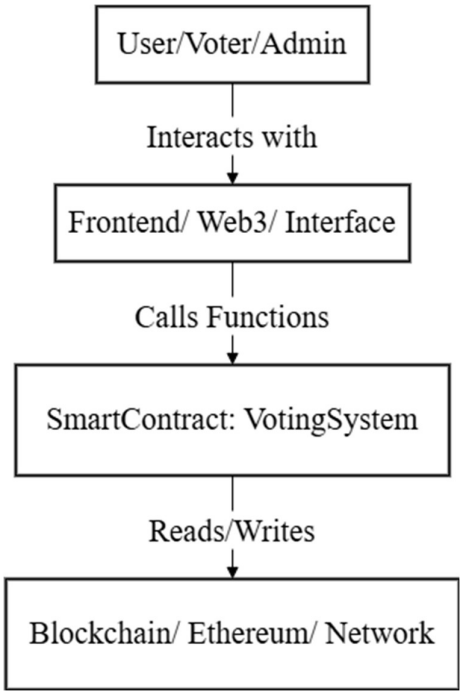

Figure 4.1 Architecture Design

Figure 4.1 illustrates the system architecture, showing how users interact with the blockchain through a smart contract. The contract, written in Solidity, handles registration, voting, and result calculation while enforcing rules and maintaining transparent, tamper-proof records on the Ethereum blockchain.

4.2 Module Description

The Decentralized Voting System is structured into three primary modules, each responsible for handling a specific aspect of the election process. These modules interact directly through the smart contract written in Solidity and deployed via Remix IDE. The

design emphasizes modularity, transparency, and tamper-resistance to ensure a reliable voting process.

4.2.1 Admin Control Module

This module grants exclusive access to the administrator (the contract deployer) for managing the election lifecycle. It ensures that only authorized users can perform sensitive operations such as registering candidates or starting and ending the voting phase.

Functions:

- addCandidate(string memory name): Registers a new candidate before voting starts.
- startVoting(): Initiates the voting period; cannot be reversed once triggered.
- endVoting(): Terminates the voting phase; results can only be accessed after this.

Enforces admin-only access using onlyAdmin modifier. Prevents unauthorized manipulation of election flow.

4.2.2 Voting Module

Handles the core functionality of vote casting while enforcing integrity rules such as one-person-one-vote. It ensures that votes are recorded immutably and only during the active voting period.

Functions:

- vote(uint candidateId): Allows a user to vote for a candidate by ID.

Prevents double voting through hasVoted mapping. Validates candidate existence. Only allows voting between startVoting() and endVoting().

4.2.3 Result Computation Module

Determines the election outcome by analyzing the votes stored on the blockchain. It includes logic to detect a tie (draw) scenario if multiple candidates receive the same highest number of votes.

Functions:

- getWinner(): Computes and returns the name of the winning candidate or “Draw” in the event of a tie.

- getVoteCount(uint candidateId): Returns the vote count of a specific candidate.

Read-only functions for result transparency. Accessible after voting has ended. Ensures automated and error-free result calculation.

4.3 Data Flow Diagram:

The DFD shows how data moves through the system, depicting interactions between external entities, processes, data stores, and flows to produce the output.

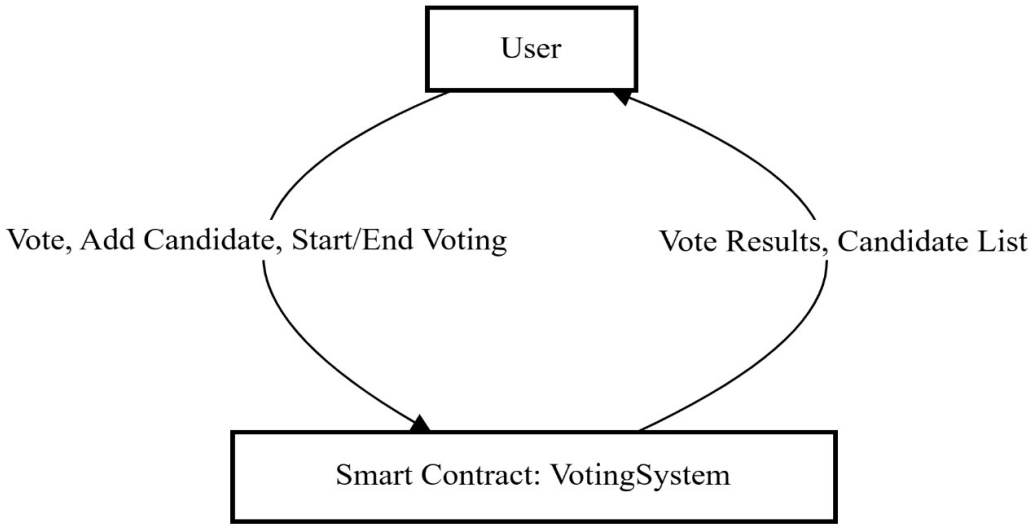

Figure 4.2 Level 0 DFD

Figure 4.2 shows a Level 0 DFD (context diagram) of the Voting System as a single process: the VotingSystem smart contract. It illustrates data flows between Users (voters and admins) and the smart contract deployed on the blockchain. Users send inputs such as votes, candidate additions, and voting start/end commands. The smart contract responds with vote results and candidate lists. This diagram simplifies the system's core interaction and emphasizes secure, transparent election management on the blockchain.

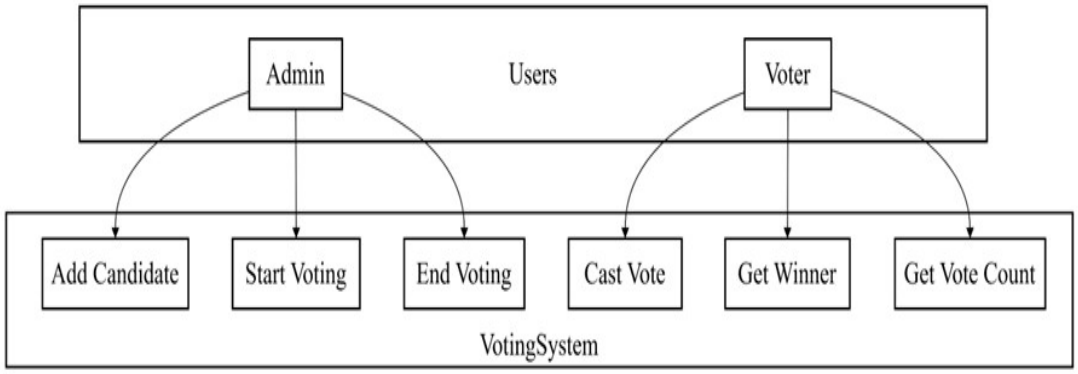

Figure 4.3 Level 1 DFD

Figure 4.3 depicts the key processes of the Voting System and data flow between Admin, Voter, and internal system functions. The Admin manages the election by adding candidates and controlling the voting period, while Voters cast votes and view results. Both roles are grouped as Users, interacting with the system through control and data inputs. Arrows indicate data direction, clarifying how users engage with the system. This diagram helps visualize the system's structure and ensures clear understanding of control and data flows.

4.4 Use case Diagram:

A Use Case Diagram is a UML tool that visually represents a system's functional requirements by showing how users (actors) interact with system functions (use cases). Figure 4.4 models interactions in the Voting System between two actors:

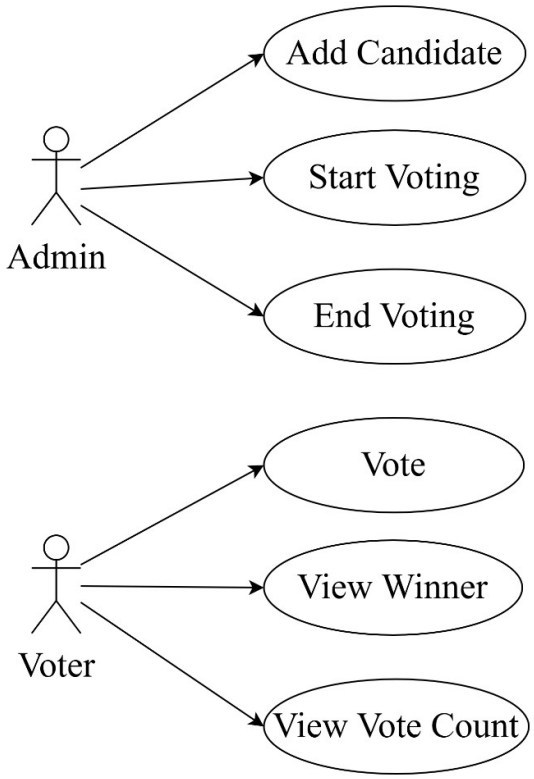

Figure 4.4 Use case Diagram

Admin: Manages voting setup and control, including adding candidates, starting, and ending voting. Voter: Participates by casting votes and viewing results, including checking the winner and vote counts. The diagram clearly distinguishes Admin's control roles from Voter's participation roles, effectively illustrating role-based interactions within a secure, organized voting system.

4.5 Class diagram

A Class Diagram is a static structure diagram in UML (Unified Modeling Language) that describes the classes, their attributes, methods, and the relationships between them. It is used to model the blueprint of object-oriented systems.

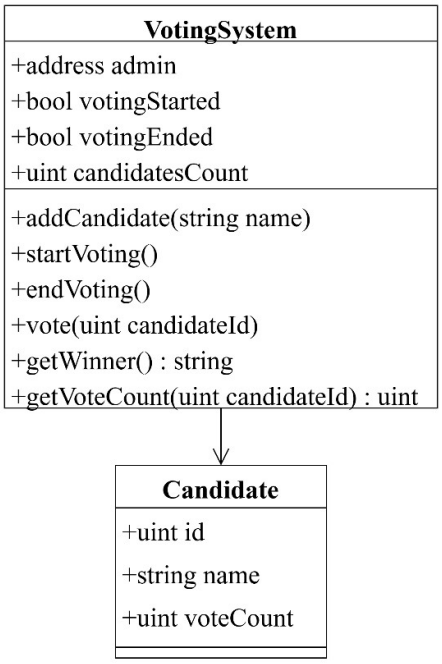

Figure 4.5 Class Diagram

Figure 4.5 presents a class diagram for a blockchain-based Voting System with two main classes. VotingSystem: Core logic managing voting, with attributes like admin address, voting status flags, and candidate count. Key methods include adding candidates, starting/ending voting, casting votes, and retrieving winners or vote counts. Candidate: Represents election candidates with ID, name, and vote count. The VotingSystem class references multiple Candidate instances, illustrating how it manages candidates and voting flow. The diagram's design suits smart contract implementation in Solidity or similar platforms.

4.6 Sequence diagram

Figure 4.6 is a sequence diagram that models the interaction between different entities (objects or actors) in a blockchain-based voting system. It shows how messages are passed between these entities over time, specifically for a simple voting scenario.

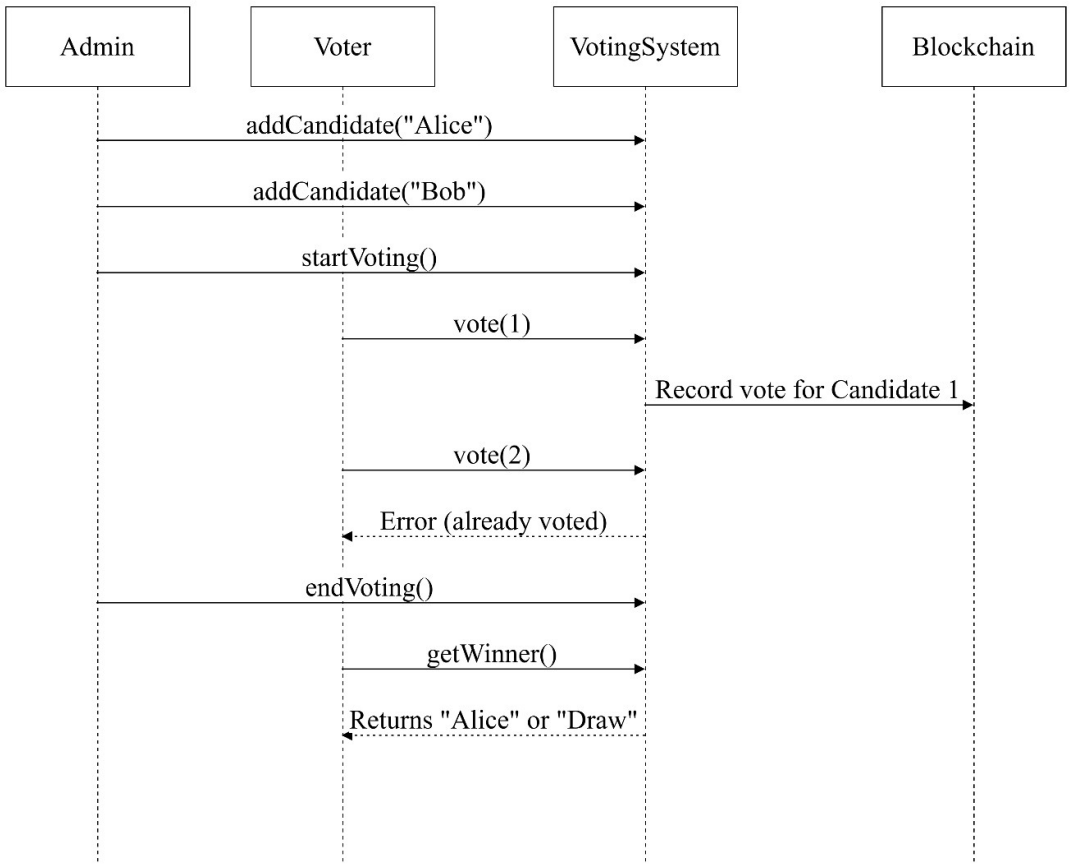

Figure 4.6 Sequence Diagram

Participants:

- Admin: Sets up the voting process.
- Voter: Casts votes.
- VotingSystem: Manages voting logic.
- Blockchain: Securely records votes.

Sequence:

1. Admin adds candidates ("Alice," "Bob").
2. Admin starts voting.
3. Voter votes for Alice; vote recorded on blockchain.
4. Voter attempts a second vote; system rejects it.
5. Admin ends voting.
6. Winner is requested; system returns "Alice" or "Draw."

This outlines the control and data flow in a blockchain-based election.

Chapter 5

Implementation

This chapter explains how the Decentralized Voting System was developed, detailing the key algorithms implemented in the smart contract and how they function. Each function is written in Solidity and deployed on an Ethereum-compatible blockchain using Remix IDE. The voting contract enforces rules like one-person-one-vote, admin-only controls, and result computation with draw detection.

5.1 Register Candidate Pseudocode

This function allows the administrator to register a new candidate. Each candidate has a unique ID, name, and an initial vote count of zero. Candidate registration is only allowed before voting starts.

Pseudocode:

```
FUNCTION addCandidate(name)
   REQUIRE voting has not started
   INCREMENT candidateCount
   CREATE new Candidate with:
      id = candidateCount
      name = input name
      voteCount = 0
   STORE in candidates mapping
END FUNCTION
```

This function ensures that no candidates can be added once voting begins. It auto-generates a candidate ID and securely stores the candidate in the blockchain mapping.

5.2 Start and End Voting Pseudocode

Only the admin can start or end the voting session, and these actions are protected to avoid misuse.

Pseudocode:

```
FUNCTION startVoting()
   REQUIRE caller is admin
   REQUIRE voting has not already started
   SET votingStarted = true
```

```
END FUNCTION

FUNCTION endVoting()
   REQUIRE caller is admin
   REQUIRE voting has started and not already ended
   SET votingEnded = true
END FUNCTION
```

These control functions manage the lifecycle of an election. Voting can only start once and can only be ended after it starts.

5.3 Cast Vote Pseudocode

This is the core function of the voting process. It records a user's vote and ensures no one votes more than once.

Pseudocode:

```
FUNCTION vote(candidateId)
   REQUIRE votingStarted is true
   REQUIRE votingEnded is false
   REQUIRE sender has not voted before
   REQUIRE candidateId is valid
   INCREMENT candidate's voteCount
   SET hasVoted[sender] = true
END FUNCTION
```

The vote is mapped to the voter's Ethereum address, preventing duplicate votes. Transactions are signed and submitted via VM.

5.4 Get Winner Pseudocode

This function is called after voting ends to determine the winner. If multiple candidates receive the same highest number of votes, the result is declared a draw.

Pseudocode:

```
FUNCTION getWinner()
   REQUIRE votingEnded is true
   maxVotes = 0
```

```
    winnerId = 0
    drawCount = 0

    FOR each candidate in candidates
        IF candidate.voteCount > maxVotes
            maxVotes = candidate.voteCount
            winnerId = candidate.id
            drawCount = 1
        ELSE IF candidate.voteCount == maxVotes
            INCREMENT drawCount
        END IF
    END FOR

    IF drawCount > 1
        RETURN "Draw"
    ELSE
        RETURN candidates[winnerId].name
END FUNCTION
```

This function scans all candidates to find the highest vote count and determine whether a tie occurred. It is a view function and does not require gas.

5.5 Get Vote Count Pseudocode

Used to retrieve the vote count for a specific candidate without modifying state.

Pseudocode:

```
FUNCTION getVoteCount(candidateId)
    REQUIRE candidateId is valid
    RETURN candidates[candidateId].voteCount
END FUNCTION
```

This function helps display real-time vote statistics on the frontend and is essential for transparency.

Chapter 6

Testing

Test cases verify the Voting System's functionality and security. They ensure all key processes work correctly. Table 6.1 describes the test case of Voting System Smart Contract

Table 6.1 Test case of Voting System

SI. No.	Function	Test Description	Input	Expected Output	Result
1	addCandidate()	Verify that only the contract owner can add a candidate.	Candidate name (e.g., "Alice")	Candidate added successfully; candidatesCount incremented.	Pass
2	addCandidate()	Ensure non-owner cannot add a candidate.	Candidate name from non-owner address	Transaction reverted with error: ―Only the contract owner can perform this action.	Pass
3	vote()	Allow a voter to vote once for a valid candidate.	Voter address + candidate ID	Vote recorded; candidate voteCount incremented; hasVoted updated.	Pass
4	vote()	Prevent double voting from the same address.	Same voter address voting again	Transaction reverted with error: ―You have already voted.	Pass
5	getCandidate()	Retrieve candidate details by ID.	Valid candidate ID	Returns candidate ID, name, and vote count.	Pass
6	hasVoted()	Check if an address has already voted.	Voter address	Returns true if voted; false otherwise.	Pass
7	candidatesCount	Verify the total number of registered candidates.	N/A	Returns the current count of candidates.	Pass

Chapter 7

Result and Discussion

This chapter presents and analyzes the results obtained from testing the Decentralized Voting System. The primary functionalities: candidate registration, voting, vote tracking, and winner calculation were all tested using Remix IDE. Screenshots from these tests validate that the system operates as intended, and the analysis explores both successes and current limitations.

7.1 Candidate Registration

We can observe the in Figure 7.1 and Figure 7.2 the Smart contract was deployed and the administrator successfully registered multiple candidates before voting started. The system prevented any candidate from being added after the startVoting() function was called.

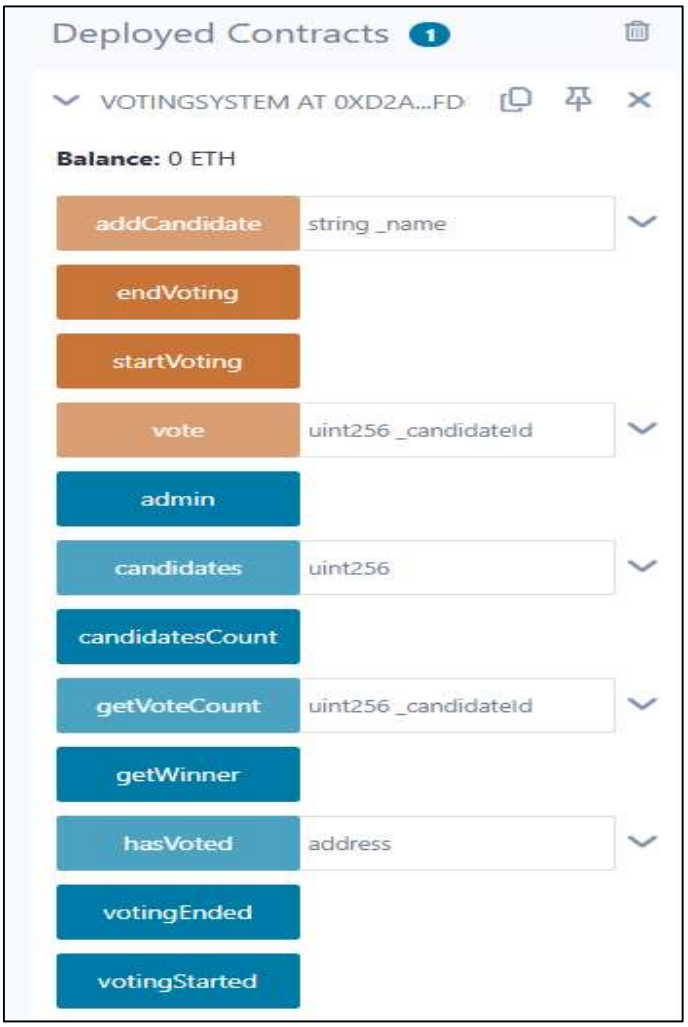

Figure 7.1 Smart Contract Deployed

Figure 7.2 Added candidates by admin

- Candidate data (ID, name, and initial vote count) was correctly stored in the blockchain mapping.
- The addCandidate() function was accessible only by the contract owner.

This ensures election integrity by locking the candidate list prior to voting and restricting access to authorized personnel only.

7.2 Voting Functionality

We can see that in Figure 7.3 users were able to vote only once. Each vote increased the selected candidate's vote count. Attempts to vote multiple times or vote after the election ended were blocked.

Figure 7.3 Users voting candidate only once

- Invalid candidate IDs or duplicate vote attempts triggered smart contract rejections.

- Transactions were confirmed on VM, and state changes were verifiable via events.

This mechanism eliminates voter fraud and guarantees a one-person-one-vote policy which is crucial for public trust.

7.3 Winner Calculation

In Figure 7.4, after voting ended, the getWinner() function correctly determined the winner based on the highest number of votes.

Figure 7.4 Winner is viewed

In cases where two or more candidates received the same number of votes, the function returned "Draw" as expected.

- The function could be called without gas fees since it's a view-only operation.
- Voters could check the winner immediately after voting closed.

The built-in draw detection avoids ambiguity and reflects real-world voting logic. No manual vote tallying was required.

7.4 Vote Count Query

Users were able to view vote counts for each candidate by inputting their candidate ID. This function was accessible both during and after voting.

- Data was fetched directly from the blockchain.

Public visibility into vote counts enhances transparency, a major benefit of using blockchain in governance.

Figure 7.5 Getting vote count of a candidate

7.5 Discussion

1. **Transparency and Immutability:** All actions (voting, registering, result computation) are publicly verifiable on-chain. Once cast, a vote cannot be changed or erased.
2. **Security:** Only the admin can control sensitive functions like starting or ending elections.
3. **Efficiency:** Transactions are executed and reflected immediately within the blockchain environment.
4. **User Experience:** The interface is simple and intuitive, ensuring easy interaction with the smart contract.

The results validate that the Decentralized Voting System achieves secure, tamper-proof, and transparent elections. Every component from candidate registration to result declaration works as intended, demonstrating blockchain's potential in digital governance. While the system excels in transparency, security, and ease of use, limitations like scalability and voter anonymity offer areas for future improvement.

Chapter 8

Conclusion

This project successfully demonstrates the implementation of a decentralized voting system using Ethereum smart contracts deployed on a private blockchain network. By using VM the system provides a secure, transparent, and tamper-proof platform for electronic voting. It ensures that each voter can cast only one vote, and that all voting data is immutably recorded on the blockchain, eliminating common issues such as vote duplication and manipulation. The project also showcases the real-world applicability of blockchain in digital governance by simulating the complete end-to-end workflow from candidate registration to vote result visualization. While the system currently operates on a local test environment, it lays the foundation for scalable and secure e-voting solutions that can be adapted for educational institutions, organizations, or community-based elections. The successful integration of backend blockchain logic with a user-friendly interface highlights the practical potential of decentralized applications in solving real-world problems.

References

[1] Noizat, P. (2017). "Blockchain Electronic Voting: How to Ensure Privacy and Transparency." *Ledger*, 2, 1–13.

[2] Ayed, A. B. (2017). "A Conceptual Secure Blockchain-Based Electronic Voting System." *International Journal of Network Security & Its Applications (IJNSA)*, 9(3), 1–9

[3] McCorry, P., Shahandashti, S. F., & Hao, F. (2017). "A Smart Contract for Boardroom Voting with Maximum Voter Privacy." *International Conference on Financial Cryptography and Data Security.*

[4] Hjalmarsson, A., Hreiðarsson, G. K., Hamdaqa, M., & Hjálmtýsson, G. (2018). "Blockchain-Based E-Voting System." *IEEE International Conference on Cloud Computing Technology and Science.*

[5] Al-Khateeb, J. H., Fawwaz, N. A., & Abdulhameed, S. (2020). "Secure and Transparent E-voting System Using Ethereum Blockchain." *International Journal of Advanced Computer Science and Applications (IJACSA)*, 11(9), 579–586.