Chapter 1

Introduction

The project titled "Election System Using Blockchain" focuses on creating a secure, transparent, and tamper-resistant election mechanism by leveraging blockchain technology. It addresses critical challenges associated with traditional voting systems such as fraud, vote tampering, and lack of transparency. Through the use of Ethereum and smart contracts, the system ensures that voting processes are decentralized, immutable, and verifiable by all participants. This project illustrates how decentralized applications (dApps) can foster trustless and robust election systems.

1.1 Overview of the Project

This project, titled "Election System Using Blockchain", leverages blockchain technology to develop a secure and transparent voting platform. Utilizing Ethereum smart contracts, the system automates vote management and guarantees that votes are irreversibly recorded without the need for a central authority. Development is carried out using Ganache for a local test blockchain, Remix IDE for smart contract authoring and deployment, and MetaMask for user interaction.

1.2 Problem Statement

Conventional voting methods are often susceptible to manipulation, data breaches, and lack of transparency. The centralization of election data can lead to vote tampering, counting errors, and diminished public trust. There is an urgent demand for an election system that ensures data integrity, eliminates unauthorized access, and delivers verifiable and real-time results.

1.3 Objectives of the Project

The primary objectives of this project are:

1. To design and develop a decentralized election system using blockchain smart contracts.
2. To guarantee the immutability of votes and prevent duplicate voting.
3. To offer a transparent, tamper-proof, and auditable voting process.
4. To provide a user-friendly voting interface using MetaMask and Remix IDE.
5. To validate the system through a controlled test environment using Ganache.

1.4 Scope of the Project

This project focuses on the development of a proof-of-concept decentralized voting platform. It

includes smart contract design, deployment on a local Ethereum network, and user interactions through MetaMask. The scope excludes deployment on a live public network and advanced voter authentication mechanisms beyond Ethereum account verification.

1.5 Methodology

The development of the Election System Using Blockchain was carried out using the Agile Software Development Methodology, specifically following the Scrum framework.

The methodology involves:

a. Requirement analysis and system design.

b. Smart contract development in Solidity.

c. Local network setup using Ganache.

d. Contract compilation and deployment via Remix IDE.

e. User interaction testing with MetaMask.

f. System validation and performance evaluation.

1.6 Summary

Chapter 1 provided an overview of the decentralized voting system project, highlighted the key problems in traditional voting, and specified the objectives, scope, and methodology. Subsequent chapters will cover literature review, system design, implementation, results, and future work

Chapter 2

Literature Survey

The rise of blockchain technology has enabled a new wave of innovation in e-governance and e-voting systems. Numerous researchers have explored its application in elections:

2.1 Introduction to Literature Review

This chapter reviews previous research and existing systems related to electronic voting and blockchain technology. It focuses on the evolution of e-voting mechanisms, the challenges faced in centralized systems, and the emerging role of decentralized platforms, especially those leveraging Ethereum smart contracts. Key academic contributions and experimental systems are examined to provide context for the design and implementation of the decentralized voting system proposed in this project.

Decentralized Voting Platform Based on Ethereum Blockchain [1] The article by Khoury et al. (2018) presents a working prototype of a decentralized voting system implemented using Ethereum smart contracts. The platform allows registered users to vote through their blockchain addresses, where votes are immutably stored and publicly auditable. The system was deployed on a local test network using Ganache, with contract logic developed via Solidity in Remix IDE, and interactions handled through a MetaMask wallet interface. The authors emphasize the elimination of centralized vote counting authorities, thereby reducing the risk of manipulation. However, the study noted potential challenges in gas costs and user onboarding complexity, especially for non-technical participants, suggesting a need for frontend abstraction and optimization in larger-scale deployments.

Towards Secure **E-Voting Using Ethereum Blockchain [2] In this 2018 IEEE study, Koç et al.** explore the privacy challenges in Ethereum-based e-voting systems and propose a solution using linkable ring signatures to anonymize voter identities. Their design ensures that votes are verifiable without being traceable to individual voters, maintaining both transparency and voter privacy. The smart contract is coded to accept blinded votes, and a reveal phase later tallies valid ballots without exposing voter information. Tested on Remix and Ganache, the model successfully blocked double-voting and maintained privacy integrity under simulated attacks. The study concludes that decentralized voting systems can achieve both end-to-end verifiability and anonymity through cryptographic enhancements, though performance trade-offs exist due to increased computational overhead.

Blockchain-Based Electronic Voting with Enhanced Security [3] Pawar et al. (2019) present a secure voting framework leveraging the Ethereum blockchain, aimed at small-scale elections such as campus voting or organizational polls. The framework includes a registration module, voting module, and result declaration—all embedded in smart contracts. Voters register using their Ethereum addresses, and the system verifies and records votes immutably. The platform uses SHA-256 hashing for ballot integrity and restricts each address to a single vote. Experiments using Ganache show real-time response with minimal latency and full traceability. While the model offers transparency and auditability, the authors highlight the absence of advanced voter verification (such as biometric checks) and recommend integrating national digital identity systems in future iterations for real-world deployment.

2.2 Existing Systems

Current electronic voting (e-voting) systems generally fall into two categories: centralized web-based platforms and early-stage decentralized blockchain prototypes. Centralized systems are often deployed by governments or private institutions using proprietary software and databases, where votes are collected, stored, and tallied by a central server. These systems can be efficient in handling large-scale elections but are highly susceptible to risks such as data breaches, vote tampering, DDoS attacks, and lack of transparency in the vote counting process.

Blockchain-based e-voting systems - particularly those built on Ethereum - aim to address these concerns by decentralizing control and ensuring immutable, auditable records. Notable implementations include those using smart contracts to restrict duplicate votes and wallet-based authentication via platforms like MetaMask. These systems are still largely at the prototype level and tested in private Ethereum environments (e.g., Ganache), with few real-world deployments to date.

2.3 Gaps in Existing Solutions

- Despite their promise, existing decentralized voting platforms suffer from several limitations:
- Scalability Issues: Most smart contract-based voting systems are tested on private testnets (e.g., Ganache) and fail to scale efficiently on Ethereum mainnet due to high gas fees and latency under network congestion.
- Voter Privacy: Basic implementations lack advanced cryptographic protections (e.g., zero-knowledge proofs or ring signatures), leading to potential vote traceability from wallet addresses.

- User Accessibility: Current systems rely heavily on technical tools like Remix IDE and MetaMask, which are not user-friendly for non-technical voters.
- Lack of Regulatory Integration: No formal integration exists with government ID systems or national electoral laws, making it infeasible for official deployment.
- Security Trade-offs: While smart contracts are transparent, they are also publicly visible and prone to exploits if not coded securely. Bugs in smart contracts could lead to irrevocable vote loss or manipulation.
- Limited Real-world Testing: Very few of these systems have been used in real elections, limiting their empirical validation in high-stakes environments.

2.4 Summary of Literature Review

The literature reviewed underscores the transformative potential of blockchain in improving transparency, auditability, and trust in voting systems. The research articles by Khoury et al. , Koç et al, and Pawar et al. showcase early-stage Ethereum-based voting prototypes with features such as one-vote enforcement, on-chain auditability, and decentralized result computation.

However, each system presents challenges in scalability, usability, and privacy. While cryptographic enhancements and hybrid on/off-chain models improve security, they increase complexity. The need for user-friendly interfaces, secure smart contract design, and regulatory integration is evident across all studies. These gaps form the motivation for the current project, which aims to implement a decentralized voting system that is secure, transparent, user-accessible, and tested under a complete development lifecycle using Ganache, Remix IDE, and MetaMask.

Chapter 3

Software and Hardware Requirements

A reliable and efficient implementation of the decentralized voting system requires a carefully selected combination of software tools and hardware components. This chapter outlines the key software and hardware requirements necessary for the development, deployment, and testing of the voting system. It covers the essential platforms, frameworks, and system specifications needed to ensure smooth operation and optimal performance during the project lifecycle.

3.1 Software Requirements

The decentralized voting system requires the following software components for development, deployment, and testing:

Table 3.1 Software requirement

Software Component	Description and Version
Solidity	Programming language used for writing smart contracts. Tested with Solidity version 0.8.
Remix IDE	Online IDE for smart contract development and testing.
Ganache	Local Ethereum blockchain simulator for development and testing.
MetaMask	Ethereum wallet browser extension used for interacting with the smart contract.
Node.js and NPM	Required for managing dependencies and running development scripts.

Web3.js	JavaScript library for interacting with the Ethereum blockchain from the front end.
Operating System	Windows 10/11 or macOS/Linux, compatible with the above tools.
Google Chrome/Firefox	Browser supporting MetaMask extension and Remix IDE.
Text Editor	Visual Studio Code or Sublime Text for writing and managing Solidity and web files.

3.2 Hardware Requirements

The successful development and testing of the decentralized voting system required basic hardware configurations capable of running blockchain simulations and supporting development tools.

Table 3.2 Hardware Requirements

Hardware Component	Minimum Specifications
Processor	Intel i5 / AMD Ryzen 5 or higher
RAM	Minimum 8 GB
Storage	Minimum 20 GB free space for software installations and blockchain data
Graphics	Integrated or discrete GPU capable of running modern browsers and IDEs
Network Connectivity	Stable internet connection for Remix IDE and MetaMask
Display	13” or larger display with a minimum resolution of 1366x768
Input Devices	Keyboard and mouse for development and testing

3.3 Summary

This chapter outlined the necessary software and hardware components required for the development and testing of the decentralized voting system. The software stack combines **Solidity**, **Remix IDE**, **Ganache**, and **MetaMask** for blockchain interactions, while the hardware requirements ensure smooth development and execution. With these specifications met, the system can be implemented effectively for prototyping and demonstration.

Chapter 4

System Design

This chapter outlines the overall design of the decentralized voting system, detailing its architecture and operational workflow. The system integrates Ethereum smart contracts, a local blockchain for development, and user interactions through wallet interfaces. The design ensures transparency, security, and scalability in recording and tallying votes. It encompasses both the backend logic and user-facing components, enabling seamless operation of the voting process.

4.1 Architecture Overview

The proposed Decentralized Voting System uses a modular architecture combining blockchain-based backend logic with a user-interactive front-end interface. The primary components of the system are:

a) Ethereum Smart Contract

Developed in Solidity, the smart contract defines and enforces all election logic, including voter registration, vote casting, candidate management, and result tallying. It resides on the Ethereum Virtual Machine (EVM), providing transparency and immutability.

b) Ganache Blockchain (Local Testnet)

Ganache provides a personal Ethereum blockchain for rapid development and testing. It allows users to simulate transactions and inspect state changes in real time without gas costs or delays, replicating a real Ethereum environment in a local setup.

c) Remix IDE

The smart contract is written, compiled, and deployed using Remix, an open-source browser-based IDE specifically tailored for Ethereum development. Remix provides debugging tools, compiler warnings, and real-time transaction feedback.

d) MetaMask Wallet

MetaMask is used as the interface for users to interact with the deployed contract. It facilitates signing transactions from registered Ethereum addresses and confirms each operation (e.g., vote submission) on the blockchain.

e) User Flow

- Voter Registration: The admin whitelists eligible addresses.

- Vote Casting: Registered users connect via MetaMask and cast votes.
- Result Viewing: At the end of the election, results can be accessed transparently from the blockchain.

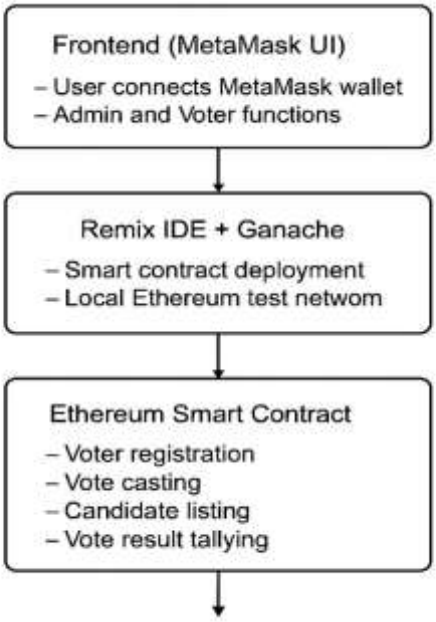

Figure 4.1 System Architecture diagram

Above showing Figure 4.1 explains layered architecture ensures decentralization, security, auditability, and extensibility.

4.2 Smart Contract Workflow

The workflow of the voting system is governed entirely by the deployed smart contract, ensuring full automation and transparency. Key logical components and execution flow are as follows:

a) Initialization

The administrator deploys the contract, adds candidate names, and sets the voting period (start and end time).

b) Voter Registration

Voters are registered via their Ethereum wallet addresses. A mapping (address => bool) is used to check eligibility.

c) Voting Logic

When a registered voter submits a vote:

- The contract checks if the sender is eligible and hasn't voted yet.
- It marks the voter as "voted" in a mapping.

- It increments the vote count for the selected candidate.
- Smart contract pseudo-code

```
function vote(uint candidateId) public {

require(eligible[msg.sender], "Not eligible");
require(!voted[msg.sender], "Already voted");
voted[msg.sender] = true;
candidates[candidateId].voteCount++;

}
```

d) Vote Integrity

- Double-voting is prevented via a voted flag.
- The vote is stored on-chain and immutable.
- Each transaction emits a blockchain event (VoteCast) to allow front-end tracking.

e) Result Calculation

Once the election ends, any user can call a public view function like getResults() that returns real-time tallies per candidate. The system is transparent—results are directly readable from the blockchain and cannot be modified.

All voting operations are gas-metered, but in the local Ganache testnet environment, execution is fast and cost-free, ideal for prototype validation.

4.3 data flow diagram

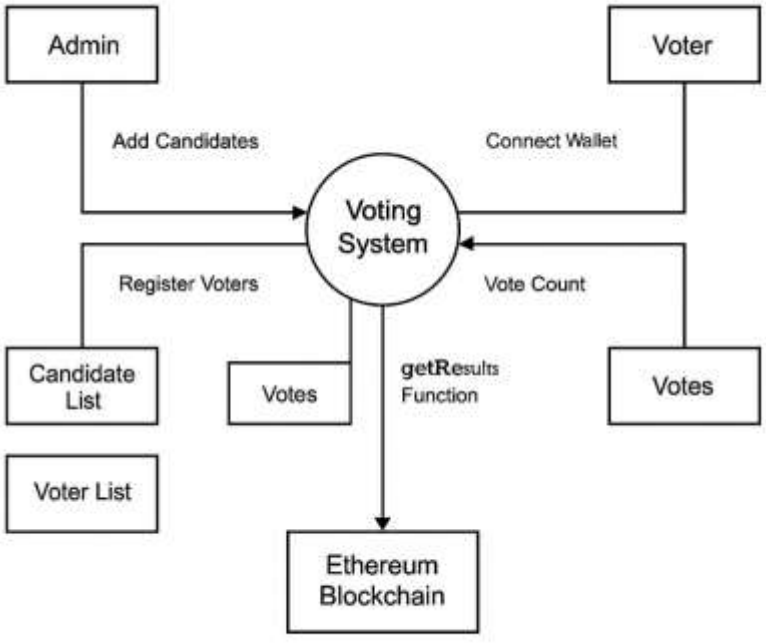

Figure 4.2 data flow diagram

The above Figure 4.3 illustrates a blockchain-based voting system using the Ethereum blockchain. The system includes Admin and Voter roles. Admins can add candidates and register voters, generating candidate and voter lists. Voters connect their wallets, cast votes, and the system counts these votes. All votes are securely recorded on the Ethereum blockchain using a getResults function for transparent result retrieval. This decentralized approach ensures tamper-proof, verifiable, and anonymous voting. It enhances trust and integrity in the election process by eliminating centralized control, preventing fraud, and enabling secure, transparent access to election outcomes via smart contracts on the Ethereum network.

4.4 Use Case Diagram

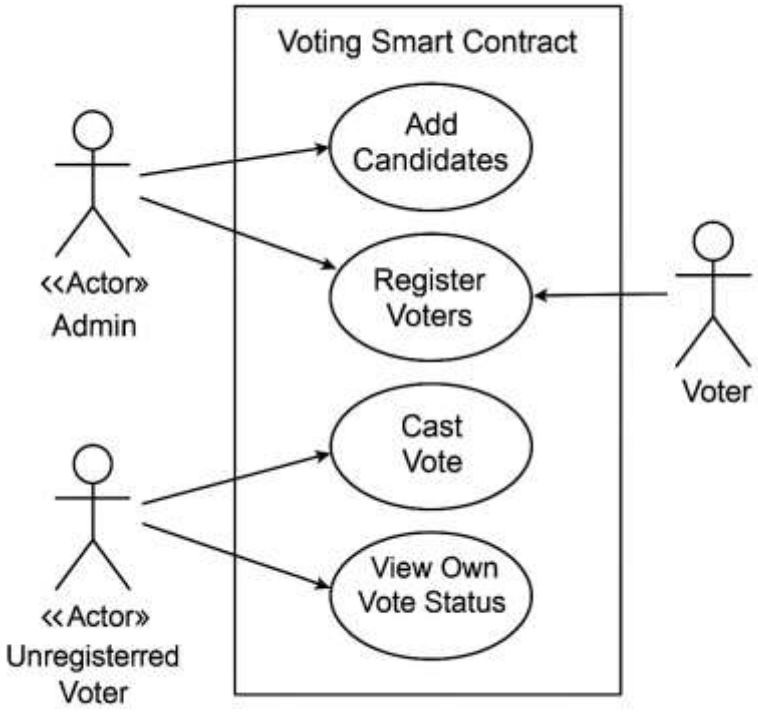

Figure 4.3 Use Case Diagram

This Figure 4.4 represents a use case model for a blockchain-based Voting Smart Contract system. It illustrates interactions between three main actors: Admin, Unregistered Voter, and Voter. The Admin is responsible for adding candidates and registering voters. Unregistered Voters can view their vote status and, once registered by the admin, can become Voters who are eligible to cast votes. The system includes four core functionalities: Add Candidates, Register Voters, Cast Vote, and View Own Vote Status. Each actor interacts with the smart contract depending on their role, demonstrating a clear access control mechanism within a decentralized voting environment.

4.5 Sequence Diagram

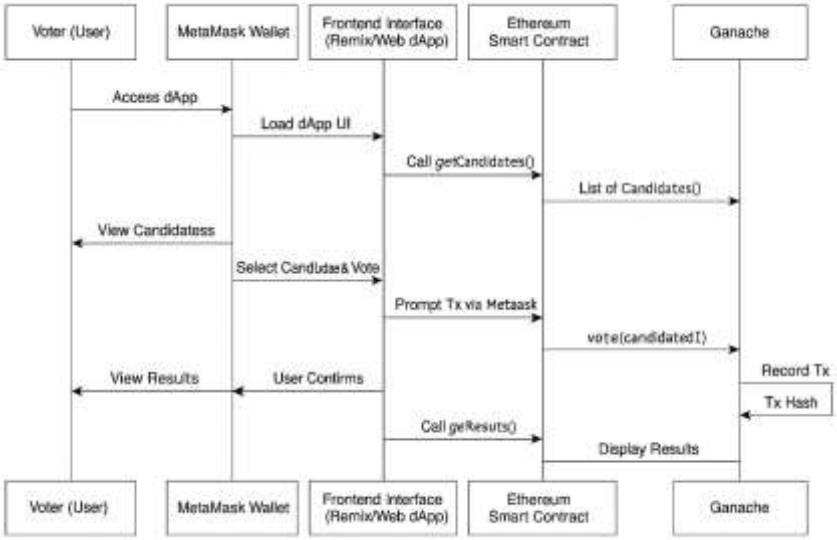

Figure 4.4 Sequence Diagram

The Figure 4.4 shows a sequence diagram of a blockchain-based voting dApp using Ethereum. It involves five components: Voter (User), MetaMask Wallet, Frontend Interface (Web or Remix dApp), Ethereum Smart Contract, and Ganache. The process starts when the voter accesses the dApp, prompting MetaMask to load the interface. The frontend then calls getCandidates() from the smart contract to fetch a list of candidates from Ganache, which is shown to the user. After selecting a candidate, the voter confirms the transaction in MetaMask. The smart contract receives the vote via vote(candidateId), and Ganache records the transaction, returning a transaction hash. Finally, the frontend calls getResults() to retrieve and display voting results. Some spelling errors are present in the diagram, such as "Candluae," "Metaask," and "geResuts()" which should be corrected.

Chapter 5

Implementation

The contract was developed using Solidity. The voting logic was tested and deployed using Remix IDE, with Ganache providing the Ethereum network for transactions. MetaMask enabled wallet integration, signing, and authorization.

5.1 Tools and Technologies Used

The implementation of the decentralized voting system uses the following tools and frameworks:

- **Solidity**: The programming language used to write the smart contract logic deployed on the Ethereum blockchain.
- **Remix IDE**: An online development environment for writing, compiling, and deploying smart contracts. It provides an intuitive interface and powerful debugging tools.
- **Ganache**: A local Ethereum blockchain simulator that allows testing without using real ETH or incurring gas costs. It provides accounts preloaded with test ETH.
- **MetaMask**: A browser extension wallet used for managing Ethereum accounts and signing transactions securely during interactions with the smart contract.

5.2 Smart Contract Implementation

The smart contract was developed using Solidity and contains the following main components:

- **Structs** for defining candidates and voters.
- **Mappings** to track registration status, votes, and prevent double voting.
- **Functions** to register voters, allow them to vote, and retrieve results.

5.3 Deployment Process

1. **Compile the Contract**: The contract is written and compiled in Remix IDE.
2. **Connect Ganache and MetaMask**: Ganache provides a local network and MetaMask is connected to use one of the test accounts.
3. **Deploy Smart Contract**: The contract is deployed to the Ganache network using Remix.
4. **Register Voters**: The admin registers voters by calling the registerVoter function with their wallet addresses.
5. **Add Candidates**: Candidate names are added using the addCandidate function.
6. **Voting**: Registered voters use MetaMask to interact with the vote function.

7. **View Results**: Any user can call the getResult function to view vote counts.

5.4 Testing Scenarios

- **Valid Voter Voting**: A registered user was able to cast a vote successfully.
- **Double Voting Attempt**: The system blocked a second voting attempt from the same address.
- **Unregistered Voter Attempt**: Unregistered addresses were denied access to vote.
- **Real-Time Results**: Admins could view real-time vote counts using public view functions.

Chapter 6

Testing

This chapter describes the testing activities carried out to validate the functionality, performance, and security of the **Decentralized Voting System Using Smart Contracts on Ethereum**. Given the project's one-month development timeline, a combination of manual and automated testing methods was used to ensure the system met its objectives. Testing was conducted in a simulated environment using **Ganache**, **Remix IDE**, and **MetaMask** to closely replicate real-world interactions.

6.1 Types of Testing Conducted

To ensure the correctness and reliability of the decentralized voting system, multiple levels of testing were performed during the one-month development period.

6.1.1 Unit Testing

Individual smart contract functions were tested in isolation using **Remix IDE's JavaScript VM**. Functions like registerVoter(), vote(), and getResult() were validated for correctness, handling expected and boundary conditions.

6.1.2 Integration Testing

Interactions between smart contracts and the **MetaMask wallet** were tested to ensure seamless transaction signing, correct state updates, and proper event handling. MetaMask's injected provider was used for transaction validation.

6.1.3 Functional Testing

The system's core workflows were tested end-to-end:

- Voter registration by the admin
- Candidate registration
- Voting by registered users
- Result tallying and public verification
- Test cases ensured only authorized actions were allowed, preventing double voting and unauthorized access.

6.1.4 Security Testing

While formal security audits were not feasible within the timeframe, common vulnerabilities were manually checked:

- **Reentrancy**: Functions were written to avoid nested calls.
- **Access Control**: Admin-only functions were restricted via msg.sender checks.
- **Input Validation**: User inputs were validated to prevent injection or overflow attacks.

6.1.5 Usability Testing

The system was tested for user experience using **MetaMask** and **Remix IDE**. Special attention was paid to clarity of error messages, transaction confirmations, and ease of navigation for non-technical users.

6.2 Testing Tools Used

- Remix IDE Debugger: To step through transactions and inspect state changes.
- Ganache Logs: To track transaction details and identify issues.
- MetaMask Activity: To monitor user interactions and confirm signed transactions.

6.3 Key Functions Tested

Table 6.1 : Key functions testing

Function	Test Result
Voter Registration	Successfully restricted to admin
Vote Casting	Allowed only for registered users
Double Voting	Blocked second vote attempt
Unregistered Voting	Automatically rejected
Real-time Vote Display	Accurate and consistent

6.4 Summary

This chapter outlined the testing strategy employed for the decentralized voting system. A combination of unit, integration, functional, security, and usability tests ensured the system performed as intended within a simulated blockchain environment. While the testing was sufficient for a proof-of-concept, future work should include formal audits, public network testing, and comprehensive security evaluations.

Chapter 7

Results and Discussion

This chapter presents the testing outcomes and evaluates the operational behavior of the decentralized voting system built on the Ethereum blockchain. The system was validated in a controlled environment using Ganache and Remix IDE, simulating a real-world blockchain network. Key functionalities including voter registration, vote casting, and result computation were analyzed for correctness, performance, and security. The discussion highlights the advantages, limitations, and potential areas for improvement, offering insights into how the system aligns with the project's objectives.

7.1 Functional Validation

The decentralized voting system was successfully implemented and tested on a local blockchain network using **Ganache**. The primary goal was to ensure transparency, prevent vote tampering, and enforce one vote per user. The system performed all intended functionalities correctly.This confirms that Ethereum smart contracts can be effectively used for transparent, fair, and secure elections. However, scalability and voter privacy remain challenges for public deployment.

7.2 Usability and Interaction

MetaMask was used by all users to connect their wallets, approve transactions, and interact with the smart contract on Remix IDE. Despite its technical nature, MetaMask provided a secure and user-friendly interface for signing votes. Gas simulation in Ganache allowed smooth interactions without transaction fees.

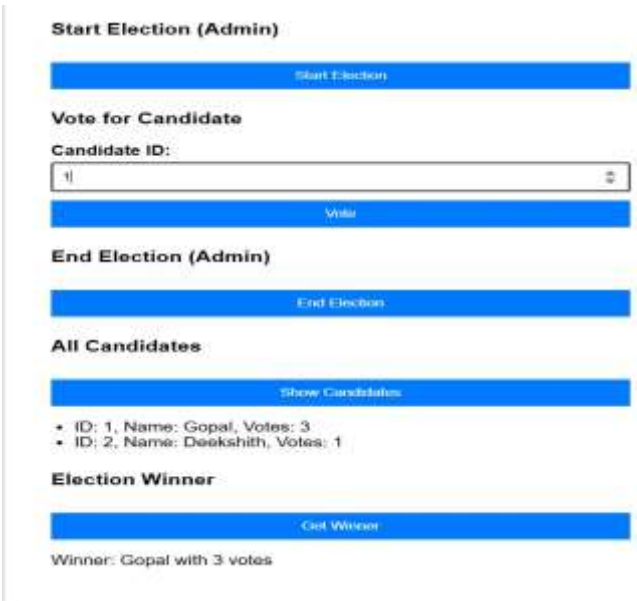

Figure 7.1 User Interface

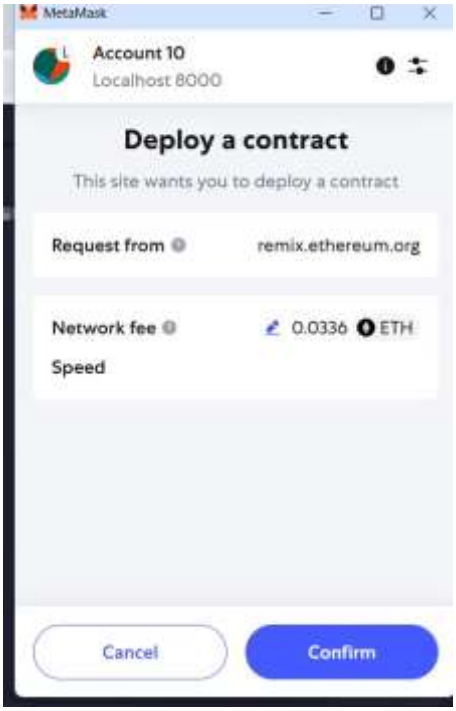

Figure 7.2 Configuration of Meta Mask to deploy a Smart Contract

7.3 Advantages Observed

- **Immutability**: All votes, once cast, were permanently stored on the blockchain.
- **Transparency**: Anyone could view vote tallies using the contract's public functions.
- **Security**: Only pre-approved Ethereum addresses were allowed to vote.
- **No Double Voting**: The system used mapping flags to track voter status and restrict multiple attempts.
- **No Manual Intervention**: Results were computed automatically through smart contract logic.

7.4 Performance Analysis

Though performance was not the primary focus, the following observations were noted during testing:

7.1 Performance Analysis

Parameter	**Value (Ganache)**
Deployment Time	~5 seconds
Average Vote Latency	~1 second

Transaction Cost	0 (simulated)
Result Query Time	< 1 second

The system performed consistently with low latency in all operations. Real-world public Ethereum networks would introduce gas fees and slightly higher delay, which can be optimized using Layer 2 solutions like zkRollups.

7.8 Summary

This chapter analyzed the system's test results and operational behavior. The voting process was secure, fair, and tamper-proof under local blockchain conditions. The discussion also highlighted the system's potential and limitations, offering insights into areas for future improvement.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

The primary objective of this project was to design and implement a decentralized voting system using **Ethereum smart contracts**, providing a **secure, transparent, and tamper-proof** electoral mechanism. The system effectively addresses many of the inherent limitations of traditional voting systems, including centralization, lack of transparency, and susceptibility to manipulation.

The solution was built using Solidity for smart contract development, deployed and tested using **Remix IDE**, **Ganache** for a local blockchain environment, and **MetaMask** as the wallet interface. The system enforced **one-vote-per-user** through mapping checks, allowed for **real-time tallying**, and maintained a fully auditable trail of voting transactions on the blockchain.

Functional and usability testing demonstrated that:

- Only registered users were allowed to vote.
- Double voting was successfully blocked.
- All results were visible and verifiable in real time.
- The system was user-interactive through the MetaMask interface.

This project confirms that blockchain technology has the potential to **revolutionize e-voting** by making it decentralized, secure, and verifiable without the need for a trusted third party. It lays the groundwork for future systems that can operate on national or institutional levels with appropriate enhancements.

8.2 Future Work

Although the current prototype performs as intended within a simulated environment, several enhancements are required to transition to a production-level, publicly accessible solution. Future developments could include:

a) Scalability and Deployment

Deploying the system on public Ethereum or Layer 2 solutions (e.g., Polygon, zkRollups) will improve scalability, reduce gas fees, and support larger user bases.

b) Enhanced Authentication and Privacy

Incorporating national ID systems, biometrics, OTP-based verification, and privacy-preserving technologies like zero-knowledge proofs and ring signatures will enhance security and user anonymity.

c) User-Friendly Interface

Developing a comprehensive web or mobile frontend will simplify the user experience, making the system accessible to non-technical participants and reducing reliance on MetaMask and Remix.

d) Comprehensive Security Testing

Implementing formal security audits, code verification, and bug bounty programs will strengthen contract integrity and mitigate potential exploits.

e) Offline and Hybrid Voting

Designing hybrid systems that allow offline data collection with subsequent blockchain synchronization can improve accessibility in low-connectivity areas and enhance overall system robustness.

References

[1] Ayed, A. B. (2017). A conceptual secure blockchain-based electronic voting system. International Journal of Network Security & Its Applications, 9(3), 01-09.

[2] 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. Springer.

[3] Anjomshoae, S., & Shamszaman, Z. (2018). Blockchain-based e-voting protocol. IEEE Access..