Mini-Project Report

On

“Decentralized E-Voting System

Submitted By

Om Jagzap

Sachin Jaiswal

Vishal kambale

Under the guidance of

Ms. Pratiksha Gujar

DEPARTMENT OF COMPUTER ENGINEERING

Matoshri College of Engineering and Research Centre, Nashik

SAVITRIBAI PHULE UNIVERSITY, NASHIK

Academic Year [2025 – 2026]

INDEX

Introduction

A decentralized application (dApp) for electronic voting (e-voting) leverages blockchain technology and smart contracts to provide a transparent, tamper-resistant, and verifiable voting process. Traditional voting systems—whether paper-based or centralized electronic systems—face challenges such as single points of failure, potential manipulation, limited transparency, and difficulties in auditing. By contrast, a dApp built on a blockchain provides an immutable ledger where votes can be recorded securely and verified by multiple parties without relying on a central authority.

This project presents the design and implementation of a Decentralized E-Voting System dApp. The system uses smart contracts (implemented in Solidity) to define election rules, voter registration, ballot casting, and vote tallying. The front-end interacts with the blockchain via a Web3 provider (e.g., MetaMask) so that voters can cast their votes easily from a browser while transactions are recorded on a blockchain (testnet or private network). The architecture emphasizes security, privacy-preserving measures, and auditability. The dApp will demonstrate how decentralization, cryptographic verification, and transparent smart contract logic can improve trust and integrity in voting processes.

Problem Statement

Current voting systems often suffer from problems such as susceptibility to fraud, limited transparency, central points of failure, and difficulties in independent auditing. The primary problem addressed in this project is to design and implement a secure, transparent, and user-friendly e-voting platform that reduces reliance on central authorities while ensuring voter privacy and verifiability.

Key challenges include:

- Ensuring voter eligibility and preventing double voting without exposing voter identities.
- Protecting ballot confidentiality while enabling verifiable tallying.
- Defending the system against common attack vectors (replay attacks, Sybil attacks, network-level censorship).
- Providing a usable interface so that non-technical users can participate securely.

Requirement Analysis

This section lists functional and non-functional requirements, hardware and software needs, and additional setup points necessary to implement and test the dApp.

Functional Requirements

1. Voter Registration: Secure registration mechanism to add eligible voters (e.g., whitelist of addresses or off-chain verification with on-chain attestation).
2. Election Creation: Admin can create elections with metadata (title, candidates, start/end time).
3. Ballot Casting: Registered voters can cast a single vote per election through the dApp; votes are recorded on the blockchain.
4. Vote Tallying: Smart contract automatically tallies votes after election close; results are publicly readable.
5. Access Control: Only authorized accounts (administrator/commission) can perform administrative tasks.
6. Audit Trail: Immutable on-chain records for each vote transaction, enabling transparent auditability.
7. Privacy: System must avoid storing personally identifiable information on-chain; use pseudonymous addresses and optional cryptographic privacy mechanisms.

Non-Functional Requirements

1. Security: Resistance to tampering and common blockchain attack vectors.
2. Scalability: Reasonable performance for expected voter counts on chosen network; design to support layer-2 or sharding if needed.
3. Usability: Simple web UI for registration, login (wallet connect), and voting.

4. Reliability: High availability during voting window; robust error handling and transaction feedback.
5. Maintainability: Modular smart contracts and front-end code for upgrades and audits.
6. Legal & Ethical Compliance: Respect privacy laws and election regulations applicable in deployment jurisdiction.

Hardware Requirements

- Development PC/Laptop: 8 GB RAM (recommended 16 GB), modern multi-core CPU.
- Storage: ≥ 5 GB free space for node/client and development tools.
- Optional: Raspberry Pi or server to run a private blockchain node for testing.

Software Requirements

- OS: Windows 10/11, Ubuntu (20.04+), or macOS.
- Development Tools: Node.js (14+), npm/yarn, Truffle or Hardhat, Ganache (for local chain), Remix (optional), MetaMask (browser extension).
- Languages & Libraries: Solidity (smart contracts), JavaScript/TypeScript (frontend), Web3.js or Ethers.js, React.js (recommended) or plain HTML/JS.
- Blockchain Network: Ethereum testnet (Sepolia, Goerli) or private/consortium network.
- Additional: OpenSSL for cryptographic utilities, Docker (optional) for containerized deployments.

System Configuration Setup

1. Install Node.js and npm/yarn.

2. Install Truffle or Hardhat and Ganache for local blockchain testing.
3. Install MetaMask in browser for interacting with dApp.
4. Configure environment variables (RPC endpoints, private keys for test accounts).
5. Ensure Solidity compiler (solc) compatible with contract version.

System Analysis / Module Description

The dApp architecture is modular and can be divided into smart contract layer, blockchain network layer, backend (optional), and front-end. Each module is designed to be independent yet cohesive to ensure security and maintainability.

Module 1: Smart Contract Module

- Implements core election logic in Solidity.
- Responsibilities: createElection, registerVoter (whitelist), castVote, endElection, tallyVotes, getResults.
- Data Structures: Election struct (id, title, candidates, startTime, endTime, isActive), mapping of voter address to Voter struct, candidate vote counts.
- Access Control: Ownable or role-based permissions to restrict administrative functions.
- Events: Emit events for Registration, VoteCast, ElectionCreated, ElectionEnded for off-chain indexing and UI updates.
- Security considerations: input validation, check for reentrancy (though minimal), use of SafeMath or Solidity 0.8+ built-in checks, and limit gas usage.

Module 2: Blockchain Network Module

- Choice of network: public testnet (Sepolia/Goerli) for demonstration or a private/consortium network for production.
- Node Providers: Infura, Alchemy, or self-hosted node for RPC endpoints.
- Transaction Handling: Gas estimation, retry strategy, and monitoring of transaction confirmations.

Module 3: Front-End (User Interface) Module

- Built with React.js (recommended) and Web3/Ethers.js for blockchain interaction.
- Features: wallet connect (MetaMask), view active elections, registration page (if required),

ballot UI, transaction status indicators, and results page.
- UX: Clear instructions, confirmation dialogs before casting vote, and feedback for pending/confirmed transactions.

Module 4: Backend / Off-Chain Services (Optional)

- Purpose: Off-chain indexing of events, user management, analytics, and privacy-preserving voter verification (e.g., integration with government ID or KYC providers).
- Components: Node.js/Express server that listens for smart contract events and writes to a database (Postgres/MongoDB) for fast reads and dashboards.
- Security: Backend should never hold private keys for user wallets; use secure key management for administrative accounts.

Module 5: Security & Privacy Module

- Voter Privacy: Do not store personally identifiable information on-chain. Use pseudonymous wallet addresses and consider cryptographic techniques like blind signatures, ring signatures, or zk-SNARKs for stronger privacy if needed.
- Double-Voting Prevention: Enforce one-vote-per-voter via on-chain checks and whitelist mechanisms.
- Auditability: All vote transactions and key events are logged on the blockchain to aid independent audits.
- Threat Model: Consider network attacks, malicious clients, compromised admin keys, and propose mitigations (multi-sig for admin actions, rate limits, monitoring).

Motive

The motive of building a decentralized e-voting dApp is to improve trust, transparency, and resilience in electoral processes. Centralized systems often require complete trust in administrators and are susceptible to tampering; decentralization distributes trust across network participants while making tampering detectable through immutable records.

This project seeks to demonstrate how blockchain and smart contracts can be used to:

- Provide verifiable election results without exposing voter identities.
- Reduce single points of failure and increase system availability.
- Offer a publicly auditable trail of votes, enabling independent verification by stakeholders.
- Explore privacy-preserving techniques that balance transparency and ballot secrecy.

Result

This section summarizes expected and experimental results from implementing the dApp in a development environment (local Ganache or public testnet).

- Smart contract deployed successfully to a local testnet (Ganache) and/or public testnet.
- Voter registration, ballot casting, and vote tallying functions executed as expected in test scenarios.
- Transactions were visible on the blockchain and events emitted for Registration, VoteCast, and ElectionEnded.
- Example performance notes: On a local Ganache instance, transactions confirm in under 1 second; on public testnets, confirmation depends on network congestion and gas price.
- Security observations: Basic double-voting prevented; additional privacy measures (like off-chain credentials or zk techniques) are recommended for production deployments.

Sample On-chain Events (Illustrative):

Event: ElectionCreated(id=1, title='Student Council 2025', start=..., end=...)

Event: VoterRegistered(address=0xabc...)

Event: VoteCast(voter=0xabc..., candidateId=2)

Event: ElectionEnded(id=1, winnerCandidateId=2)

Conclusion

The Decentralized E-Voting dApp demonstrates the potential advantages of applying blockchain technology to electoral processes: transparency, tamper-evidence, and auditability. While the prototype shows promising improvements over centralized approaches, deploying a production-grade e-voting system requires careful consideration of legal, ethical, and technical factors—particularly voter privacy, identity verification, and resistance to large-scale attacks.

Future work includes integrating advanced privacy-preserving cryptography (e.g., zero-knowledge proofs), designing robust identity attestation mechanisms that respect voter anonymity, scaling solutions (layer-2), formal verification of smart contracts, and user-experience improvements for mass adoption.