Universitatea Politehnica București

BLOCKCHAIN-BASED ELECTRONIC VOTING SYSTEM

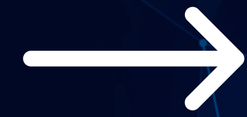
How It Works

Step 1: Voter connects MetaMask wallet to the app.

Step 2: Smart contract verifies voter eligibility.

Step 3: Voter casts a vote via the blockchain.

Step 4: Votes are tallied in real time and stored immutably.



Key Features:

- End-to-end transparency.
- Decentralized vote storage.
- Anonymous and secure participation.

Tools and Technologies

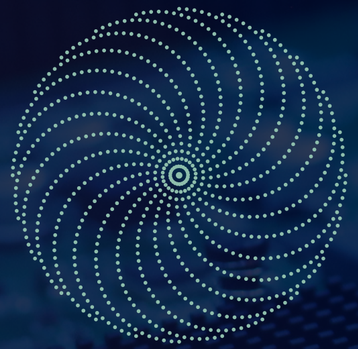
Core Technologies Used

- Blockchain Framework: Ethereum
- Smart Contracts: Solidity
- Frontend: MetaMask, Web3.js
- Backend: Node.js, TypeScript
- Wallet Integration: MetaMask for secure authentication
- Key Security Mechanisms: Public/private key encryption, tamper-proof records.

Highlights of Our Voting System

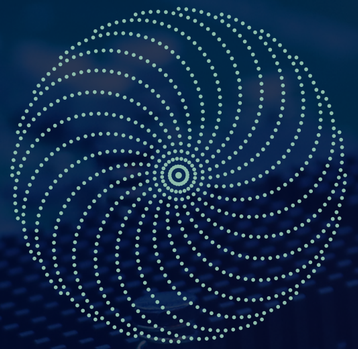
Secure Voting Process

Votes are stored immutably on the blockchain, ensuring they cannot be tampered with



Simple Transparency

Results are publicly verifiable directly on the blockchain ledger



Basic Accessibility

Anyone with a MetaMask wallet and internet connection can vote

Foundational Implementation

Built with Ethereum smart contracts, designed to demonstrate the potential of blockchain technology in voting

Demo

Core Functionalities in Code

- **Admin Management**: The contract owner (admin) can add candidates, verify voters, and control the election state
- **Candidate and Voter Data**: Stored as mappings for efficient access
- **Election State**: Tracks whether the election has started or ended

Key Functions: Start and End Election

Enables the admin to control the election lifecycle.

```
function startElection() public onlyAdmin {
    require(!start, "Election already started");
    start = true;
    end = false;
    emit ElectionStarted();
}

function endElection() public onlyAdmin {
    require(start, "Election not started");
    end = true;
    start = false;
    emit ElectionEnded();
}
```

Key Functions: Add Candidate

Allows the admin to register new candidates.

```
// Function to add a candidate to the election
function addCandidate(string memory _header, string memory _slogan) public onlyAdmin {
    candidateDetails[candidateCount] = Candidate({
        candidateId: candidateCount,
        header: _header,
        slogan: _slogan,
        voteCount: 0
    });
    emit CandidateAdded(candidateCount, _header);
    candidateCount++;
}
```

Key Functions: Register and Verify Voters

Handles voter registration and verification for election participation.

```
// Function for a user to register as a voter
function registerAsVoter(string memory _name, string memory _phone) public {
    require(!voterDetails[msg.sender].isRegistered, "Already registered");
    voterDetails[msg.sender] = Voter({
        voterAddress: msg.sender,
        name: _name,
        phone: _phone,
        weight: 1,
        voted: false,
        delegate: address(0),
        vote: 0,
        isVerified: false,
        isRegistered: true
    });
    voters.push(msg.sender);
    voterCount++;
}
```

```
// Function to verify a voter by the admin
function verifyVoter(address voterAddress, bool _verifiedStatus) public onlyAdmin {
    require(voterDetails[voterAddress].isRegistered, "Voter not registered");
    voterDetails[voterAddress].isVerified = _verifiedStatus;
}
```

Key Functions: Vote Casting

Ensures secure and authenticated voting.

```
// Function for a voter to cast a vote
function vote(uint256 candidateId) public {
    Voter storage sender = voterDetails[msg.sender];
    require(sender.weight > 0, "No right to vote");
    require(!sender.voted, "Already voted");
    require(sender.isVerified, "Not verified to vote");
    require(start, "Election not started");
    require(!end, "Election has ended");
    require(candidateId < candidateCount, "Invalid candidate ID");

    sender.voted = true;
    sender.vote = candidateId;
    candidateDetails[candidateId].voteCount += sender.weight;

    emit VoteCast(msg.sender, candidateId);
}
```

Key Functions: Determine Winner

Determines the candidate with the highest votes after the election ends.

```
// Function to compute and return the winner of the election
function getWinner() public view returns (uint256 winnerId, string memory header, string memory slogan) {
    require(end, "Election not ended");
    uint256 maxVotes = 0;
    uint256 winningCandidateId;
    bool tie = false;

    for (uint256 i = 0; i < candidateCount; i++) {
        if (candidateDetails[i].voteCount > maxVotes) {
            maxVotes = candidateDetails[i].voteCount;
            winningCandidateId = i;
            tie = false;
        } else if (candidateDetails[i].voteCount == maxVotes) {
            tie = true;
        }
    }

    require(!tie, "Tie detected, no clear winner");

    Candidate memory winner = candidateDetails[winningCandidateId];
    return (winner.candidateId, winner.header, winner.slogan);
}
```

Potential Applications

- Elections: Secure digital elections at all levels.
- Surveys and Polls: Immutable data for private organizations or public opinion studies.
- Governance: Transparent decision-making in DAOs (Decentralized Autonomous Organizations).
- Key Impact: Increased trust in the voting process, enhanced accessibility, and global scalability

THANK YOU