BLOCKCHAIN BASED VOTING SYSTEM

A decentralized voting application built with Next.js frontend and Hardhat blockchain backend.

Developed By

- Niza Khunga (2021503321)
- Faith Mayani (2021538354)
- Enala Saishi (2021463711)

About Blockvote

Blockvote is a secure, transparent voting system that leverages blockchain technology to ensure election integrity. Votes are recorded on the blockchain, making them immutable and verifiable by anyone.

Technology Stack

- Frontend: Next.js 15 with React 19
- Backend: Hardhat development framework
- Blockchain: Ethereum-based smart contracts
- Wallet Integration: MetaMask
- State Management: React Context API
- UI Framework: React Bootstrap

Prerequisites

Before installing Blockvote, you need to have these installed on your computer:

1. Ganache: https://archive.trufflesuite.com/ganache/ - Local blockchain for testing
2. Python: https://www.python.org/downloads/ - Programming language
3. Git: https://git-scm.com/ - Version control system
4. Node.js https://nodejs.org/en - JavaScript runtime (v18 or higher)
5. VS Code: https://code.visualstudio.com/download or any code editor
6. MetaMask https://metamask.io/ - Crypto wallet browser extension

Installation Guide

Step 1: Set up MetaMask Network

1. Open MetaMask and click on the network dropdown.

2. Select "Add network" and enter these details:

   - Network Name: Ganache
   - RPC Server: `HTTP://127.0.0.1:7545`
   - Chain ID: `1337`
   - Currency: ETH

Step 2: Download and Extract Files

1. Download the Blockvote project files.

2. Extract them to a folder on your computer (like your Desktop).

Step 3: Open Project in VS Code

1. Open VS Code.

2. Click `File > Open Folder`.

3. Navigate to the `blockvote` folder you extracted.

Step 4: Install Dependencies

1. Open the terminal in VS Code (`Terminal > New Terminal`).

2. Switch the terminal to Git Bash if available.

3. Navigate to the `blockvote` folder if you're not already there:

```
cd blockvote
```

4. Install frontend packages:

   npm install

5. Navigate to the backend folder:

   cd blockchain_backend

6. Install backend packages:

   npm install

Step 5: Configure Ganache

1. Open the Ganache application.
2. Click "New Workspace".
3. Name it "blockvote".
4. Click "Start" - keep Ganache open while using Blockvote.

Step 6: Configure Admin Addresses

1. In VS Code, open this file:

   blockchain_backend/contracts/AccessControl.sol

2. Find these lines around line 23:

   // Hardcoded admin addresses

   // Hardcoded auditor address

3. In Ganache, copy the first address (Index 0).
4. Replace the admin address in the code with your copied address.
5. Copy the second address from Ganache (Index 1).
6. Replace the auditor's address in the code with your copied address.

Step 7: Deploy Smart Contracts

1. In the terminal, make sure you're in the `blockchain_backend` folder.

2. Run this command to deploy contracts:

   npx hardhat ignition deploy ./ignition/modules/VotingModule.js --network localhost

3. Confirm with "yes" when prompted.

4. You will see output with two important addresses:

   - Voting contract address
   - Results contract address

5. Copy these addresses for the next step.

Step 8: Update Contract Addresses

1. Open this file in VS Code:

   blockvote/src/contracts/Results.json

2. Find the "address" field and replace it with your Results contract address.

3. Open this file:

   blockvote/src/contracts/Voting.json

4. Find the "address" field and replace it with your **Voting** contract address.

5. Open this file:

   blockvote/src/context/VotingContext.js

6. Find these lines:

const VOTING_CONTRACT_ADDRESS = 'old-address-here';

const RESULTS_CONTRACT_ADDRESS = 'old-address-here';

7. Replace both addresses with your new deployed contract addresses.

Step 9: Run the Application

1. In the terminal, navigate back to the root `blockvote` folder:

   ```
   cd ..
   ```

2. Start the development server:

   ```
   npm run dev
   ```

3. Open your browser and go to: `http://localhost:3000`

Step 10: Connect Wallet Accounts

1. In the Blockvote website, click "Connect Wallet".
2. In MetaMask, click "Add account" or "Import account".
3. Select "Private key".
4. In Ganache, click the key icon next to the first account (Index 0).
5. Copy the private key.
6. Paste it into MetaMask and import the account.
7. Repeat for the second account (Index 1) for the auditor role.

Code Structure Overview

Smart Contracts (Backend)

AccessControl.sol - Role-based access control system

- Defines three roles: Voter, Auditor, Admin
- Hardcoded admin and auditor address for security
- Manages role assignments and permissions
- Provides role verification functions

Voting.sol - Main voting contract

- Manages election lifecycle (creation, voting, finalization)
- Handles voter registration and vote casting
- Implements Merkle tree for voter eligibility verification
- Provides vote verification functionality

Results.sol - Results management contract

- Handles election result calculation and verification
- Provides real-time and finalized result access
- Manages winner determination and tie scenarios
- Access control for admin/auditor only functions

Frontend Structure

Context Providers (`src/context/`)

- `VotingContext.js` - Manages wallet connection and contract interactions
- `VoteContext.js` - Handles voting operations and registration
- `ElectionContext.js` - Manages election creation and management
- `ResultsContext.js` - Handles result retrieval and display
- `AdminContext.js` - Provides admin-specific functionality

Pages (`src/app/`)

- `/` - Home page with overview
- `/how-it-works` - Process explanation (updated to match actual workflow)
- `/elections` - Election listing and voting interface
- `/verify-vote` - Vote verification functionality
- `/results` - Election results display
- `/dashboard` - User dashboard
- `/faq` - Frequently asked questions

Components (`src/components/`)

- Navigation, modals, forms, and display components
- Organized by functionality (elections, results, verification, etc.)

Key Features

1. Wallet Integration - MetaMask connection with proper error handling
2. Role-Based Access - Different permissions for voters, auditors, and admins
3. Vote Verification - Both automatic and manual vote verification options
4. Real-time Results - Live updates for admins/auditors during elections
5. Transparent Process - All votes recorded on blockchain for public verification
6. Security Features - Merkle proofs, encrypted votes, and role validation

Using Blockvote

1. Connect your admin or auditor account from MetaMask.
2. As an admin, you can create new elections.
3. As an auditor, you can verify election results.
4. Voters can connect their accounts and participate in active elections.

Voting Process

1. Connect Wallet - Authenticate using MetaMask
2. Explore Elections - View available elections on the Elections page
3. Register to Vote - Complete registration for desired elections
4. Cast Vote - Select candidate and submit encrypted vote
5. Verify Vote - Use the Verify Vote page to confirm vote recording
6. View Results - Check finalized results on the Results page

Troubleshooting

- Make sure Ganache is running before starting the application.
- Ensure you've updated all contract addresses correctly.
- If transactions fail, check that you have enough ETH in your Ganache accounts.
- Check browser console for detailed error messages.
- Verify MetaMask is connected to the correct network (localhost:1337).

Support

If you encounter any issues during setup, please check that all prerequisites are installed correctly and that you've followed each step carefully.

For technical support, contact the development team with specific error messages and your setup configuration.

Development Team Contributions

Niza Khunga (2021503321)- Smart contract development and blockchain backend

- Designed and implemented AccessControl, Voting, and Results contracts
- Set up Hardhat development environment and deployment scripts
- Implemented role-based access control and election lifecycle management

Faith Mayani (2021538354) - Frontend development with Next.js

- Built responsive UI components using React Bootstrap
- Implemented page routing and user interface design
- Created election management and voting interfaces

Enala Saishi (2021463711) - Frontend-backend integration

- Developed context providers for state management
- Implemented wallet connection and contract interaction
- Created vote verification and result display functionality
- Set up contract ABIs and address configuration

This collaborative project demonstrates a complete blockchain voting system with secure, transparent, and verifiable elections using modern web technologies and Ethereum smart contracts.