DS Project Report: Kelvin Likollari, Divyank Sharma, Francesco, Yi Lu

December 9, 2024

1 Introduction

For the Distributed Systems Project, we developed a Decentralized Voting System. The goal of this project is to develop a decentralized voting system where the voting power of each voter is proportional to the number of shares they hold in a company or organization. This approach is suitable for shareholder meetings or governance decisions in decentralized organizations.

2 Application Theme

The application addresses the need for secure, transparent, and decentralized governance. Traditional voting systems often rely on centralized authorities, raising concerns about trust and security. By leveraging blockchain, our system ensures transparent, immutable, and verifiable votes without relying on a central authorities. The application helps participants by aligning voting power with shareholding, fostering fairness and enabling shareholders to influence governance. This model is ideal for shareholder meetings and decision-making in blockchain-based organizations.

3 Application Features

The application fosters many features that allow a great degree of functionality within our system, such as Weighted Voting, Share-Based Registration, Flexible Decision Thresholds, Time-Locked Shares, Limit proposal creation to shareholders with a set share percentage, Voting History Dashboard, Real-Time Voting Progress, Shareholder Rewards for Voting, and Roles for Managing Proposals and Elections.

4 Implementation

The implementation of the decentralized voting system is structured into three main components: **Backend, Frontend**, and **Smart Contracts.**

For the **Backend**, a Node.js server connects the smart contracts with the frontend, handling API calls and ensuring proper interaction with the blockchain. It also handles metadata and aggregates voting statistics for visualization.

The **User Interface** is built with Vue.js, providing an interactive dashboard for voting, proposal creation, and viewing results. It fetches data directly from the backend and interacts with Metamask wallet for secure transactions.

Smart Contracts: The core logic of the application is implemented using Solidity. The smart contracts manage and implement all the application features mentioned previously. These contracts were deployed and tested using the Remix IDE and integrated with Hardhat for local Ethereum node testing.

In this way, we ensure that we have a modular application architecture, helping in future scalability scenarios, with clear separations among backend processing, frontend interactions and smart contracts logic.

5 Technologies

The technologies used included Node.js for the backend, Vue.js for the frontend, Solidity for the smart contracts and Remix IDE to compile and deploy our contracts. We separated the project into 3 main areas: the frontend, the backend and the smart contracts implementation, so that each aspect of the application could be developed on its own, without directly interfering with the others.

6 Running the Project

To run the project, we will need 3 terminals: one for the backend, the frontend and for starting a local Ethereum network node.
For the **backend**: *cd voting-system-backend → npm run dev*
For the **frontend**: *cd voting-system-frontend → npm run serve*
For the **local ETH network**: *cd voting-system-smart-contracts → npx hardhat node*
For **compiling and deploying the contract**s, we used Remix IDE.

Pro Tip: Make sure you do npm install in every separate directory to install the required dependencies.

7 Problems Encountered

During the development of this project, we came across several errors that made the process frustrating. In particular, the most frequent error was **JSON-RPC Error** which is a generic error that can happen for different reasons, such as not fetching correctly the abi from the compiled smart contracts, running out of gas, or having logical errors in our smart contracts code. This was resurfacing again and again after being fixed for a while by using **Remix IDE** (which helps in identifying logical errors in our solidity code) and reinstalling Metamask, re-adding the network and the Metamask accounts.

8 Improvements

Improvements include adding aggregated results for anonymity, optimizing gas costs, and creating a mobile-friendly interface.