Decentralized Arbitration system

Team Kodierer

Jai Samir Gauns Dessai
Chirag Suraj Mahajan
Conrad Cleophus Alves
Sanish Laxman Pagui

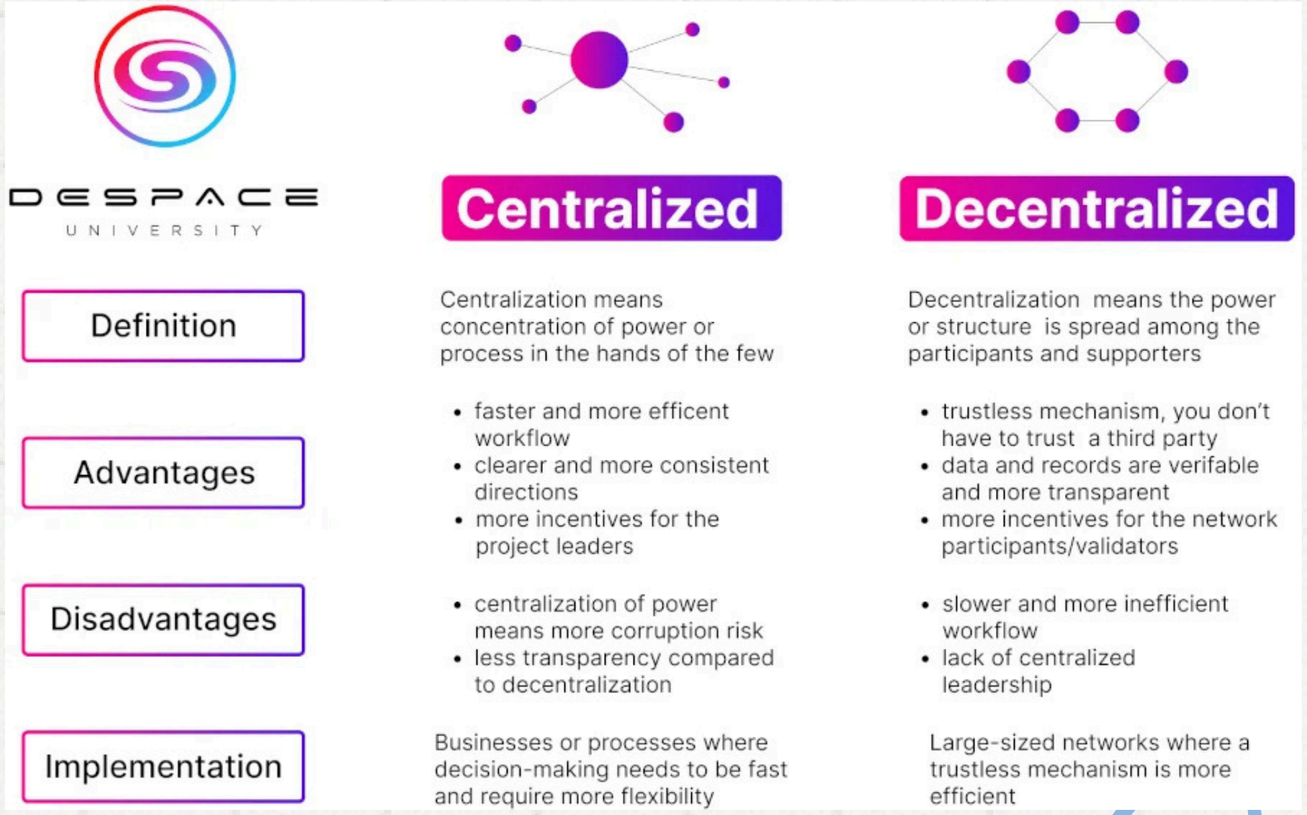
FROSCHE: Decentralized Dispute Resolution System

FROSCHE

The problem centers around the need for a fair, transparent, and decentralized dispute resolution system that reduces reliance on traditional courts. Traditional dispute mechanisms often suffer from lengthy processes, high costs, and potential biases. Our solution aims to create a blockchain-based arbitration platform that enables users to resolve disputes in a decentralized manner. By using a token-based system, we introduce incentives for fair judgment, transparency through smart contracts, and community-driven conflict resolution, ultimately providing a quicker, cost-effective, and unbiased alternative to conventional dispute resolution methods.

Solution

We developed a decentralized dispute resolution platform using blockchain technology, allowing disputes to be resolved transparently and efficiently. At its core, our solution uses a set of smart contracts to manage staking, voting, reward distribution, and juror selection, ensuring a secure, tamper-proof process. Users can stake tokens to participate in dispute resolution, vote on outcomes, and earn rewards for fair decisions. The system ensures that all actions are recorded on the blockchain, making them immutable and verifiable. This approach reduces the time, cost, and bias associated with traditional dispute mechanisms while promoting community involvement.



Project Architecture

01

Smart contracts in Solidity for mock GRULL token minting, voting system, jury selection, reward distribution, token balance checking and staking tokens

02

Frontend in NextJS for server side rendering, MetaMask for blockchain interactions and Tailwind CSS for styling the UI.

03

Backend (Ethereum Blockchain & Solidity): Deployed smart contracts on Ethereum handle dispute creation, staking, voting, juror selection, and reward distribution, ensuring decentralized and immutable transaction records.

Walkthrough of Contracts

MockGRULL (Token Minting)

This contract is responsible for minting GRULL tokens used in the system. It allows jurors to obtain tokens that are required for staking and participating in the dispute resolution process.

VotingSystem

Handles the creation of disputes and facilitates the voting process. Jurors cast their votes, and the contract collects and tallies the results to resolve the dispute based on the majority decision.

JurorSelection

Manages the decentralized selection of jurors. Jurors are chosen randomly from the pool of stakers, ensuring fairness and reducing the likelihood of manipulation by any single party.

StakingContract

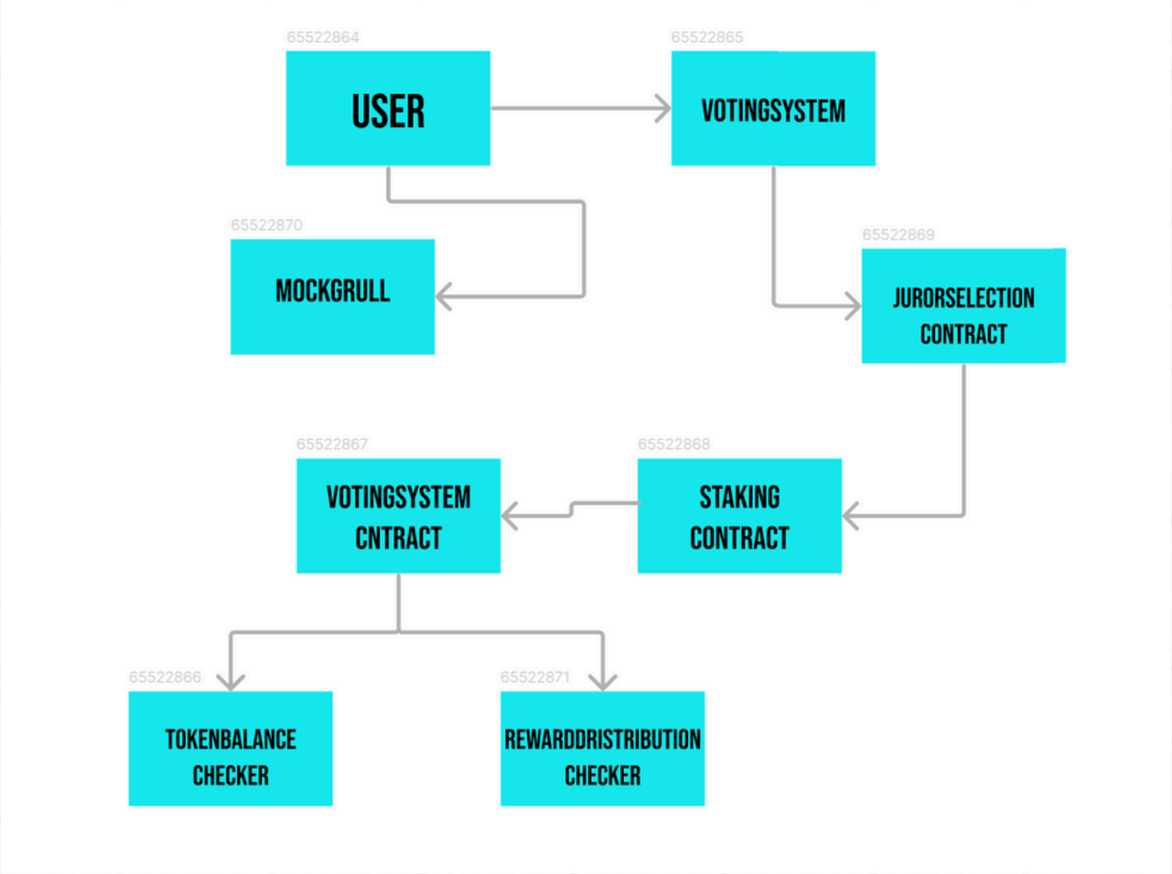
Jurors must stake GRULL tokens to be eligible for selection and voting. This contract manages the staking process and locks the tokens during the dispute resolution.

RewardDistribution

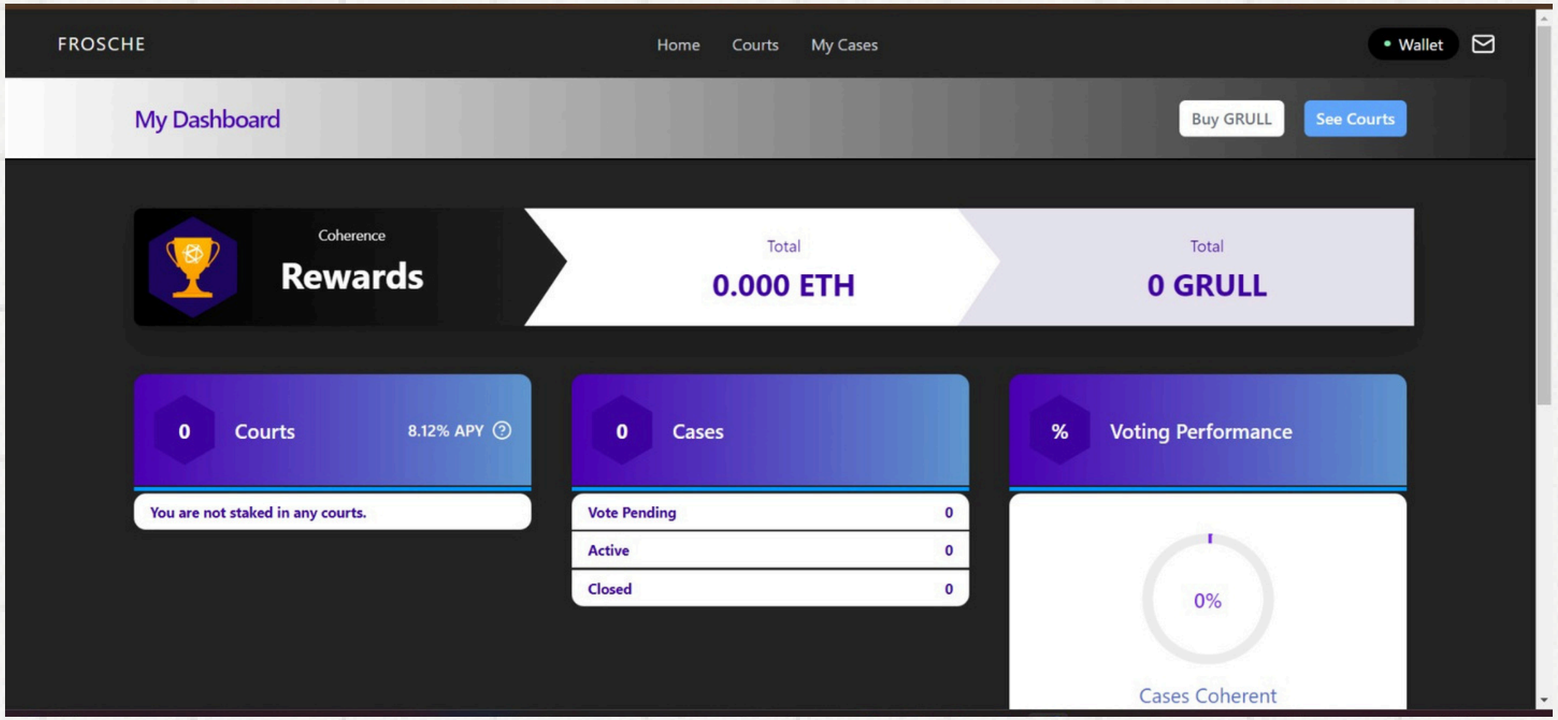
After a dispute is resolved, this contract ensures that jurors who voted with the majority are rewarded, while those who voted dishonestly may face penalties, creating an incentive for honest participation.

TokenBalanceChecker

Verifies the GRULL token balance of jurors to ensure that they meet the staking requirements before they can participate in the voting process.



Frontend Structure



Tech Stack:

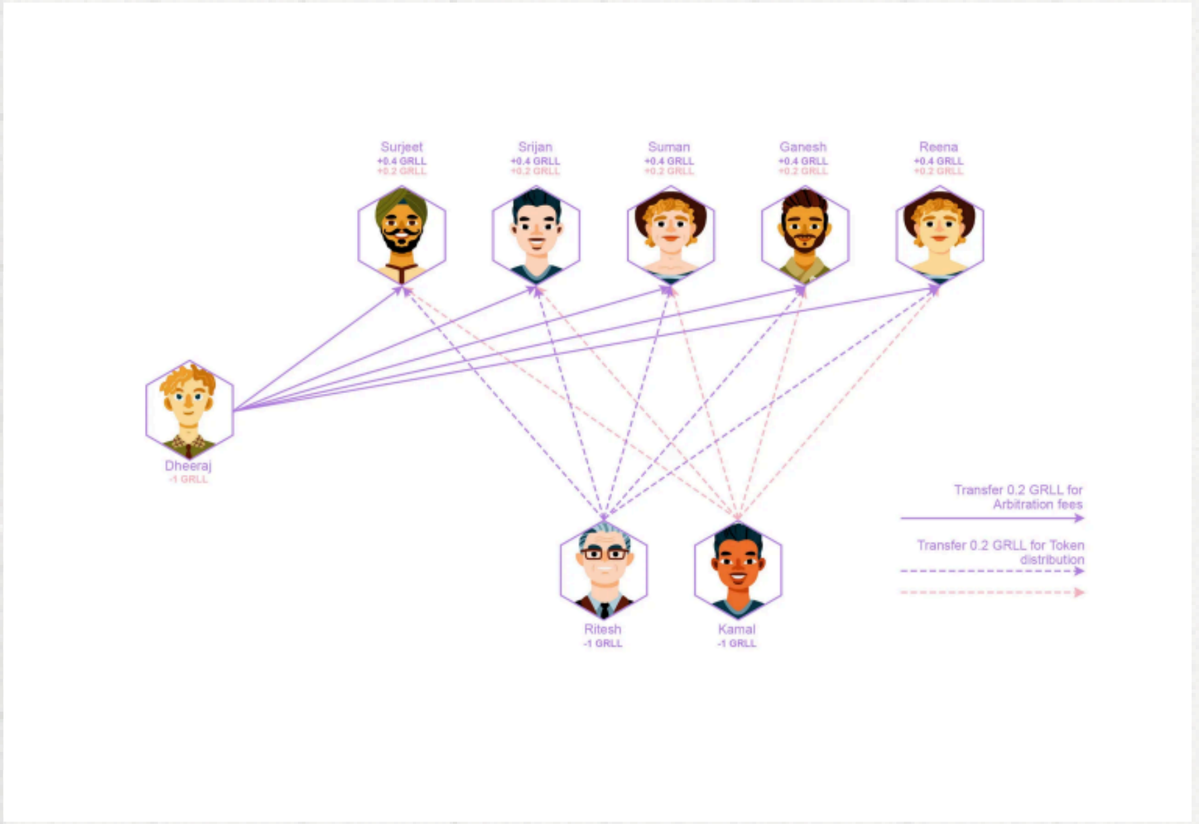
- Next.js: Used to build the frontend interface of the decentralized dispute resolution system.
- Ethers.js: For interacting with the Ethereum blockchain and smart contracts.
- Web3: Integrated for handling blockchain connections, including interactions with Metamask, Rainbowkit and Wagmi.

UI Description:
The user interface is designed to allow easy interaction with the dispute resolution system. It uses key elements , including:

- Home Page: Overview of the system and current disputes.
- Voting Page: Where jurors can view active cases and cast their votes.
- Dispute Creation Page: Allows users to submit new disputes and provide relevant details.

Backend Structure

The backend consists of five interconnected smart contracts deployed on the Ethereum blockchain, each serving a specific function within the dispute resolution ecosystem. The VotingSystem contract manages dispute creation and voting processes, while the StakingContract allows users to stake tokens for voting eligibility. The RewardDistribution contract handles reward allocation for jurors, and the JurorSelection contract facilitates the selection of jurors based on predefined criteria. This architecture ensures transparency and security, allowing for a decentralized and efficient dispute resolution process.



```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract RewardDistribution {
    IERC20 public grullToken;
    mapping(address => uint256) public rewards;
    mapping(address => uint256) public lastClaimed; // Track last reward claim time for each juror
    uint256 public claimCooldown = 1 days; // Minimum time before a juror can claim rewards again
    uint256 public maxVoteWeight = 100; // Maximum weight per vote
    uint256 public minVotesThreshold = 5; // Minimum votes required to be eligible for reward

    event RewardsDistributed(address[] jurors, uint256[] rewards);
    event RewardClaimed(address indexed juror, uint256 amount);

    /**
     * @dev Distribute rewards to jurors based on their votes.
     * @param _jurors List of juror addresses.
     * @param _votes List of votes each juror received.
     */
    function distributeRewards(address[] memory _jurors, uint256[] memory _votes) public {
        require(_jurors.length == _votes.length, "Mismatched arrays");

        uint256 totalReward = grullToken.balanceOf(address(this));
        uint256 totalVotes = 0;
        for (uint256 i = 0; i < _votes.length; i++) {
            require(_votes[i] <= maxVoteWeight, "Vote weight exceeds maximum allowed");
            totalVotes += _votes[i];
        }
```

Frontend-Backend Integration

The integration between the Next.js frontend and the Ethereum smart contracts is achieved through the use of the ethers.js library, which facilitates communication with the Ethereum blockchain. The frontend interface interacts with the smart contracts by calling their functions through web3 provider methods. Users can initiate transactions directly from the UI, such as staking tokens, voting on disputes, and resolving cases. By connecting MetaMask, users can sign transactions securely, ensuring that all interactions with the blockchain are authenticated. This integration enables real-time updates of the UI based on contract state changes, creating a seamless user experience within the decentralized dispute resolution platform.

Challenges and Learnings

- Integration Issues: Faced challenges in integrating smart contracts with the Next.js frontend, highlighting the importance of thorough testing and debugging in blockchain applications.
- Gas Efficiency: Gained insights into optimizing smart contract functions for gas efficiency, emphasizing the need to minimize transaction costs for users.
- User Experience: Learned the significance of providing clear user feedback during interactions, such as loading states and confirmation messages, to enhance the overall user experience.
- Collaboration: Improved teamwork skills by working closely with team members, sharing knowledge, and overcoming obstacles together.
- Time Management: Experienced the importance of effective time management in project development, particularly when dealing with unexpected challenges and tight deadlines.

Future Work

- Full Integration: Aim to fully integrate the frontend with the backend contracts for seamless user interactions and functionality.
- User Interface Enhancements: Plan to refine the UI/UX based on user feedback to improve usability and aesthetic appeal.
- Additional Features: Explore implementing additional features, such as multi-signature wallets and enhanced dispute resolution mechanisms, to expand the platform's capabilities.
- Testing and Deployment: Focus on thorough testing in a live environment to ensure reliability and security before official deployment.
- Community Engagement: Foster a community around the platform to gather feedback, ideas, and encourage user participation in the dispute resolution process.

Conclusion

- Innovative Solution: Our project addresses the critical need for an efficient, transparent, and decentralized dispute resolution mechanism using blockchain technology.
- Collaboration and Learning: The development process has been a significant learning experience, enhancing our skills in smart contract development, frontend technologies, and overall project management.
- Potential Impact: With further refinement and integration, our platform has the potential to revolutionize how disputes are resolved in various sectors, providing a fair and democratic approach to justice.
- Next Steps: We are committed to continuing the development of this project, incorporating feedback, and working towards a robust and user-friendly platform.
- Thank You: We appreciate your attention and welcome any questions or feedback regarding our project!