SMART DAO

Introduction

SmartDAO is a pioneering platform designed to enable individuals, especially those without technical expertise, to create and engage with Decentralized Autonomous Organizations (DAOs). By simplifying the process of DAO contract creation and interaction, SmartDAO democratizes access to blockchain governance, making it accessible and manageable for a broader audience.

Vision

To integrate "non-tech" individuals into the blockchain ecosystem, making it an inclusive and diverse space.

Mission

- Engage Non-Technical Users: Providing tools for effective participation in the blockchain ecosystem.
- Develop a No-Code Platform: Allowing users to manage DAOs without needing to write code.
- Reach the Next Billion Users: Making blockchain and DAOs accessible to a global audience.

Platform Overview

Key Features:

- User-Friendly Interface: Intuitive design for ease of use.
- Customizable Smart Contracts: Templates for various DAO structures.
- Interoperability: Seamless interaction with other DAOs and blockchain networks.
- Security and Transparency: Operations secured on the Ethereum blockchain.

Technical Overview Smart Contracts

SmartDAORouter:

Struct Definitions:

DAO: This struct appears to hold information about each DAO, including names, contract addresses (voting and treasury), and some additional metadata like logo and website URLs.

Voting: It defines the structure of a voting instance, including details like the voting name, description, choices, and end date.

TreasuryVoting: Similar to Voting, but specifically for treasury-related decisions.

Interfaces:

IVotingContractFactory and ITreasuryContractFactory: These interfaces suggest the existence of factory contracts for creating voting and treasury contracts, which aligns with the no-code approach of your platform.

IDefaultERC20Factory and IDefaultERC721Factory: These are for creating ERC20 and ERC721 tokens, indicating your platform's compatibility with these token standards.

IStorage: This interface likely relates to data storage and retrieval for DAOs.

IVotingContract: It defines methods for creating new votings, voting in them, retrieving voting data, and getting results.

Central Coordination: The presence of these interfaces and structs suggests that SmartDAORouter acts as a central coordinator. It likely handles the creation of DAOs, voting, and treasury contracts, and manages interactions between these components.

DefaultERC20

Token Attributes:

Name and Symbol: The contract stores the token's name and symbol, which are essential identifiers in the ERC20 standard.

Decimals: This is set to 18, following the common standard for ERC20 tokens.

Total Supply: The contract keeps track of the total supply of tokens.

Balance Management: balanceOf: A mapping to keep track of each address's token balance.

Transfer Function: Allows token holders to transfer tokens to another address. The function checks if the sender has enough balance before transferring.

Allowance and Approval Mechanism: allowance: This mapping tracks how much one address is allowed to spend on behalf of another.

Approve Function: Allows a token holder to set an allowance for another address, enabling them to spend a certain amount of tokens.

transferFrom Function: Enables a third party to transfer tokens from one address to another, within the limits of the allowance set by the token holder.

Events:

Transfer: Emitted when tokens are transferred, including when tokens are minted to the creator.

Approval: Emitted when a token holder sets an allowance for another address. Constructor: Initializes the contract with the given name, symbol, and total supply, and assigns the total supply of tokens to the creator's address.

DefaultERC721

Token Attributes:

- **Name and Symbol**: Similar to the ERC20 contract, these are identifiers for the NFTs.
- **Total Supply:** The contract keeps track of the total number of NFTs that can exist.

Ownership

- **_tokenOwner:** A mapping to track which address owns a specific NFT.
- **_ownedTokensCount**: A mapping to keep count of how many NFTs each address owns.
- ownerOf Function: Returns the owner of a specific NFT, ensuring the token exists.

Transfer

- **transferFrom Function:** Allows the transfer of NFTs from one address to another. It includes checks to ensure the sender owns the token and that the recipient address is valid.

Mint

- **_mint Function**: Enables the creation (minting) of new NFTs. It checks that the target address is valid, the token hasn't been minted already, and that the tokenId is within the total supply limit.

Balance

- **balanceOf Function**: Returns the number of NFTs owned by a specific address.

Events

- **Transfer**: Emitted during the transfer and minting of NFTs.
- **Approval**: Not implemented in the snippet shown, but typically used for approving others to transfer NFTs on behalf of the owner.

DefaultERC20Factory

Functionality:

- The contract contains a function createDefaultERC20, which takes parameters like the name, symbol, total supply of the token, and the creator's address.

Token Creation

- This function creates a new instance of the DefaultERC20 contract, passing the provided parameters. This effectively mints a new ERC20 token with the specified characteristics.

Returns

- After creating the new ERC20 token, the function returns the address of this newly created token contract.

DefaultERC721Factory

Functionality:

- The contract includes a function createDefaultERC721, which accepts parameters like the name, symbol, and total supply of the NFT.

NFT Creation

- This function creates a new instance of the DefaultERC721 contract with the provided parameters, effectively minting a new NFT collection.

Returns

- After the creation of the new ERC721 token, the function returns the address of this newly created NFT contract.

TreasuryContractFactory

Functionality:

- The contract includes a function createTreasuryContract, which takes two parameters: _votingOpener and _voter. These parameters likely represent the addresses that have the authority to open votings and participate in the treasury decision-making process.

Treasury Contract Creation

- This function instantiates a new TreasuryContract with the provided addresses. The TreasuryContract is likely responsible for managing the financial aspects or assets of a DAO.

Returns

- After creating the new Treasury contract, the function returns the address of this newly created contract.

VotingContractFactory

Functionality:

- The contract contains a function createVotingContract, which requires two parameters: _votingOpener and _voter. These likely denote the addresses that are authorized to initiate voting and participate in the voting process, respectively.

Voting Contract Creation

- This function instantiates a new VotingContract using the provided addresses. The VotingContract is presumably the main contract for handling voting-related activities within a DAO, including creating ballots, voting, and tallying results.

Returns

- After creating the new Voting contract, the function returns its address.

In summary, the factory contracts (DefaultERC20Factory, DefaultERC721Factory, TreasuryContractFactory, VotingContractFactory) in your SmartDAO project are integral to its no-code philosophy, providing straightforward mechanisms for users to deploy ERC20 tokens, NFTs, and essential DAO governance structures (Treasury and Voting contracts). This approach significantly lowers the barriers to entry for creating and managing decentralized organizations.

Conclusion

SmartDAO, through its intricate network of smart contracts, creates an ecosystem where ease of use, security, and efficient management are paramount. Each contract is designed not only for a specific purpose but also to synergize with others, ensuring a cohesive DAO experience. This integration is key to simplifying DAO management, making it accessible for the next billion users.

Through its sophisticated network of smart contracts, SmartDAO creates an ecosystem where each contract not only serves a specific purpose but also synergizes with others to ensure a cohesive and efficient DAO experience. This integration is pivotal for simplifying the DAO management process, making it accessible and intuitive for non-technical users.