Siu King Sum

🔗link: View the Code on GitHub
🔗link: SiuToken Token Tracker on Sepolia Etherscan

Here are two versions:
Technical Version – a detailed explanation with clear processes
Simple Version – a basic, easy-to-understand explanation

Technical Version (Clear Explanation)

SiuToken and SiuDAO

This report is intended for engineers with experience in Solidity and blockchain development. SiuToken is designed as an ERC-20 token with flexible fee mechanisms, token locking, and airdrop capabilities, and serves as the foundation for governance weight in SiuDAO. All code has been tested through multiple rounds on the Sepolia testnet and integrates seamlessly with MetaMask and Remix IDE.

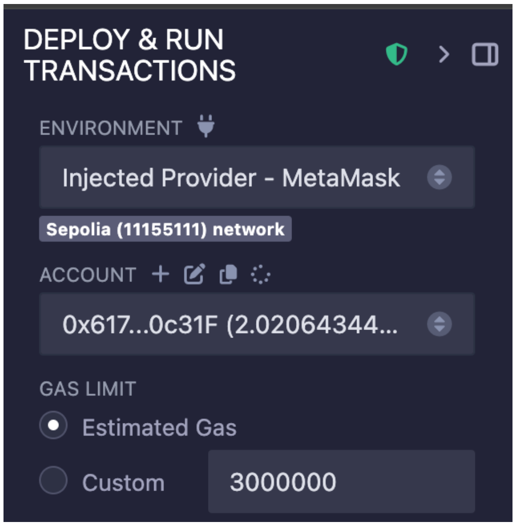
Deployment Environment

The project uses Solidity version 0.8.0, which includes built-in overflow checks to significantly reduce integer-related security risks. SiuToken inherits directly from OpenZeppelin's ERC20 implementation to ensure standard compliance. SiuDAO is implemented with custom structures for governance logic, with the SiuToken contract address injected as a dependency.

Development and debugging were conducted in Remix IDE, with deployment to the Sepolia testnet. MetaMask is configured with the Injected Provider to interact with the test network. The gas limit is set to 3,000,000, which is sufficient to cover initial minting and proposal interactions (Figure 1).

Figure 1

Remix + Sepolia + MetaMask (Gas: 3M)



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

Injected Provider - MetaMask

Sepolia (11155111) network

ACCOUNT

0x617...0c31F (2.02064344...

GAS LIMIT

● Estimated Gas

○ Custom 3000000

SiuToken Contract Design

Standard Inheritance and Initialization

The contract inherits from OpenZeppelin's ERC20. The constructor accepts an initialOwner address and mints 1 million SIU tokens (18 decimals) in a one-time issuance, ensuring a fixed total supply.

Flexible Fee Mechanism

The fee variable is stored as a percentage, currently set to 1. The transfer function overrides the default ERC20 logic by first calculating feeAmount = amount * fee / 100, then splitting the fee between two predefined addresses (40% and 60%) based on the feeRecipients mapping. The fee percentage can be dynamically adjusted using setFee(), but cannot exceed 100. The sum of all fee recipient percentages is fixed at 100.

Token Locking and Transfer Check

The lockTokens(lockPeriod) function allows holders to freeze their balance by storing block.timestamp + lockPeriod in lockTime. The transferWithLockCheck function verifies whether the current time has passed the unlock time before allowing transfers. Lock data is stored in a mapping, enabling each address to have an independent countdown.

Airdrop Batch Transfer

The airdrop() function accepts a list of addresses and a fixed transfer amount, then uses a loop to call _transfer for each recipient in a single transaction. Since the number of loop iterations grows linearly with recipients, the gas cost is **O(n)**. In production, it's recommended to use a Merkle Tree verification approach to compress multiple distributions into a single on-chain proof, significantly reducing gas usage.

Security and Optimization

The contract does not use require (amount > 0), allowing zero-amount transfers. Fee distribution is currently limited to two fixed addresses; if dynamic addition/removal of recipients is needed in the future, consider using an array with adjustable ratios via setFeeRecipients. All arithmetic is protected by overflow checks in Solidity 0.8.x. Reentrancy risks are low, as _transfer is an internal call with no external interactions. Minor gas optimizations can be achieved through unchecked { ++i; } and minimizing event emissions.

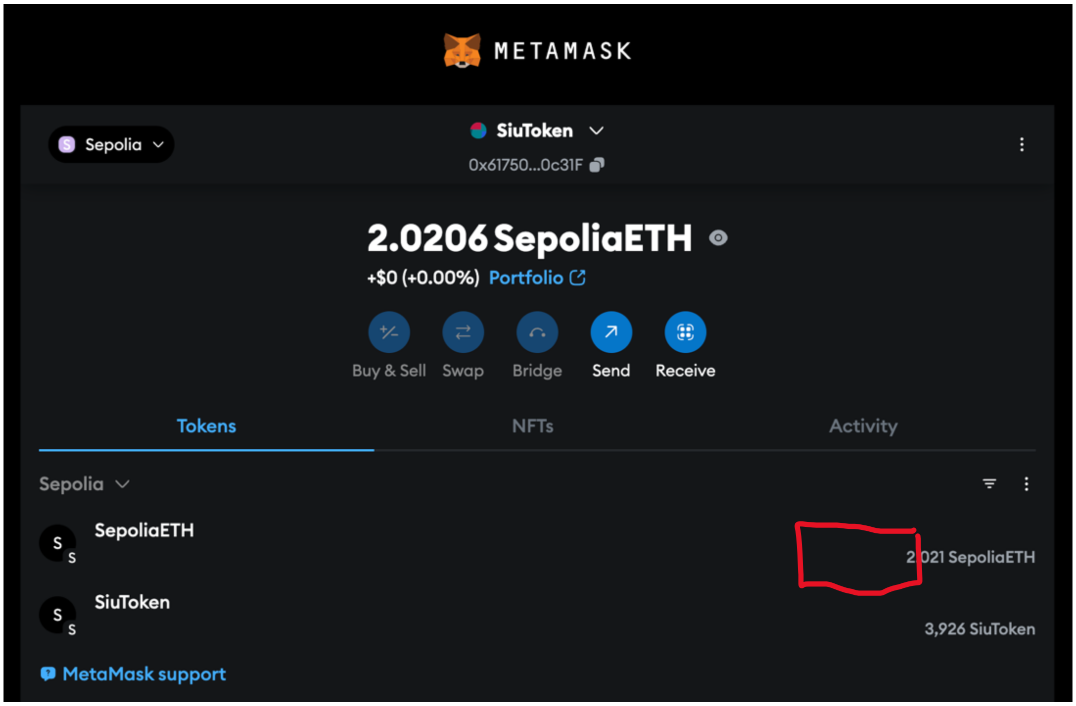
SiuDAO Contract Design

Governance Weight Binding

The governance token is the already deployed SiuToken. The SiuDAO contract receives the token's address in its constructor and casts it into a SiuToken interface for balance queries. Voting power is calculated in real-time using siuToken.balanceOf(msg.sender) during the vote() call. For example, as shown in the figure, my own MetaMask account holds only 3,926 SiuToken (Figure2), so my single vote is equal to 3,926 (Figure 3).

Figure 2

My own MetaMask account holds only 3,926 SiuToken



0x61750...0c31F

2.0206 SepoliaETH

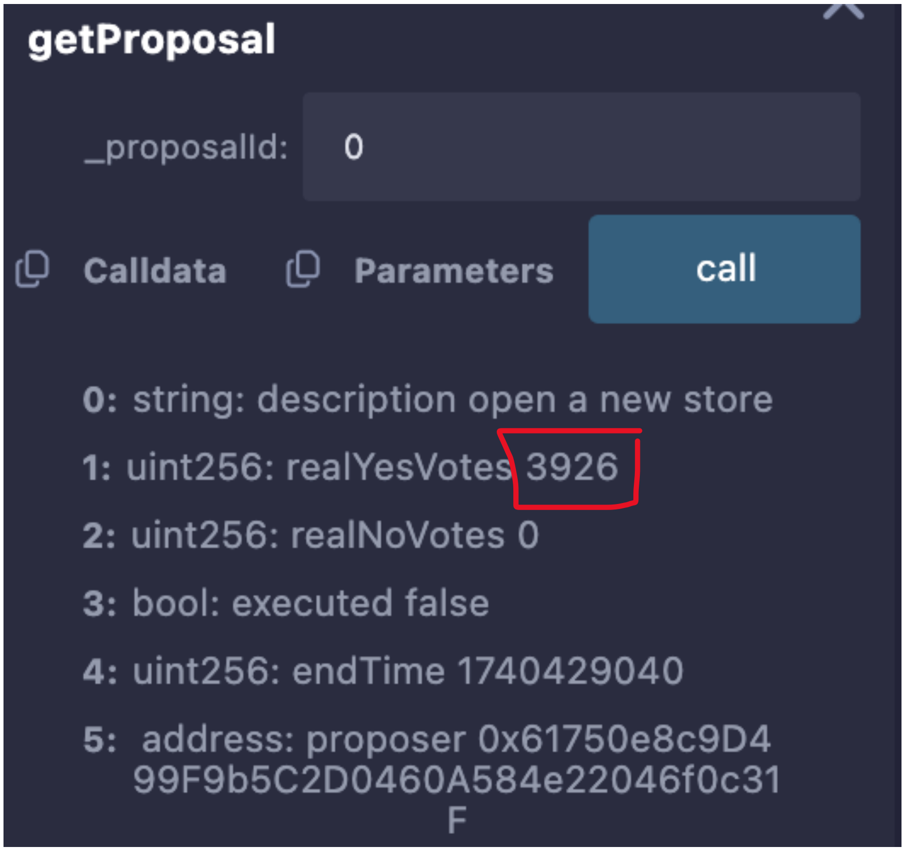
+$0 (+0.00%) Portfolio

SepoliaETH 2.021 SepoliaETH

SiuToken 3,926 SiuToken

Figure 3

My single vote is equal to 3,926



_proposalId: 0

```
0: string: description open a new store
1: uint256: realYesVotes 3926
2: uint256: realNoVotes 0
3: bool: executed false
4: uint256: endTime 1740429040
5: address: proposer 0x61750e8c9D4
   99F9b5C2D0460A584e22046f0c31
   F
```

Proposal Lifecycle

createProposal generates a Proposal struct and stores it in a mapping, using an auto-incrementing nextProposalId as the index. The proposal's endTime is determined by _votingPeriodInSeconds. The vote() function allows each address to vote once per proposal within the voting period, incrementing either the yesVotes or noVotes field based on the voter's stance. executeProposal() can be called only after the deadline to mark the proposal as executed = true, but currently does not include any asset-handling logic.

Vote Measurement and User-Friendly Interface

Since SiuToken uses 18 decimal places (1 * 10**18), the DAO internally stores vote counts in wei units. The getProposal() function returns human-readable values by dividing by 10^{18}, making it easier for frontend display. However, without a snapshot mechanism, token transfers after voting can alter the total voting power. To prevent governance manipulation, it's recommended to upgrade to OpenZeppelin's ERC20Votes or adopt the Compound-style checkpoint mechanism.

Governance and Security Risks

The contract currently lacks a quorum (minimum turnout), pass threshold, and proposal filtering, allowing potentially malicious or repetitive proposals. The executeProposal() function contains no actual business logic and serves only as a minimal working example. For production use, the following upgrades are advised:

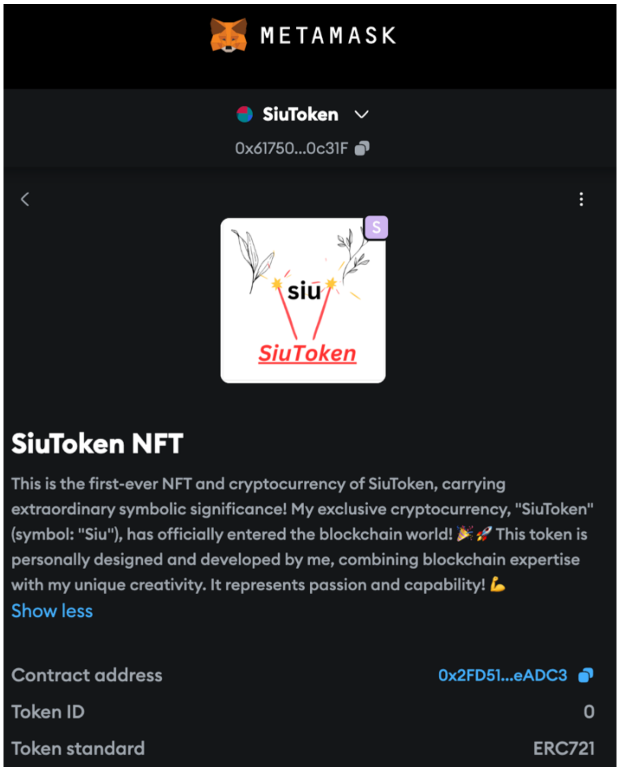
- Proposal fees or staking to deter spam proposals
- Minimum participation rates and relative or absolute majority thresholds
- Timelock mechanisms to safely control external contract calls

NFT Module Integration Plan

The project plans to introduce ERC-721 membership tokens or ERC-1155 badges to represent different governance tiers or DAO identity roles. NFTs may serve as bonus entitlements during token lock-up or as airdrop claim credentials, and can be integrated into the SiuDAO voting logic to enable a hybrid "one-person-one-vote" model**.** In fact, I recently created an NFT on OpenSea (figure 4), purely out of personal interest and without any real-world utility. However, this also highlights my diverse knowledge across different fields—beyond investment and technical skills, I also have a strong interest in blockchain technology.

Figure 4

NFT



SiuToken

0x61750...0c31F

SiuToken NFT

This is the first-ever NFT and cryptocurrency of SiuToken, carrying extraordinary symbolic significance! My exclusive cryptocurrency, "SiuToken" (symbol: "Siu"), has officially entered the blockchain world! 🎉🚀 This token is personally designed and developed by me, combining blockchain expertise with my unique creativity. It represents passion and capability! 💪

Show less

Contract address	0x2FD51...eADC3
Token ID	0
Token standard	ERC721

Testing and Deployment Process

I connect Remix to Sepolia using Injected Provider – MetaMask**.** When deploying SiuToken, the msg.sender is passed as initialOwner to receive 1,000,000 SIU. Then, SiuDAO is deployed with the SiuToken address as a parameter. Manual testing includes scenarios such as token locking, airdrops, fee adjustment, proposal creation, voting, and execution. All on-chain activities are confirmed through event logs monitored in the Remix Console. All transaction details and related information can be publicly found on Etherscan and Blockscout.

Future Improvements and Scalability

Future enhancements will include integrating ERC20Votes and Governor to upgrade the governance module, enabling features such as snapshots, proposal execution conditions, and Timelock. The project also plans to adopt UUPS or Transparent Proxy patterns to make both the token and the DAO upgradeable. Additionally, the current static fee recipient setup will be replaced with a dynamic array structure, along with interfaces for adding or removing recipients as needed. Finally, the airdrop mechanism will be refactored into a Merkle Drop system to significantly reduce gas consumption.

Conclusion

SiuToken and SiuDAO form a basic yet functional system for early community testing**.** With upgrades like snapshots, Timelock, audits, and NFT identity, it can become more secure and ready for full deployment.

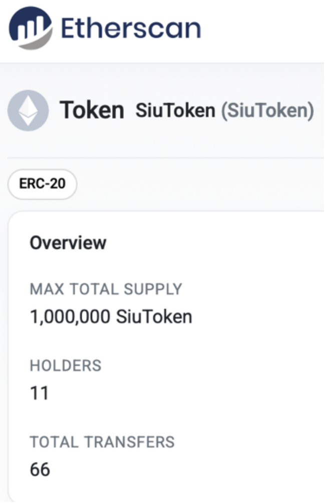
Simple Version

SiuToken and SiuDAO

This project involved the development and deployment of an ERC20-standard virtual currency smart contract, "SiuToken," on the Sepolia testnet (Figure 1). The contract was designed with the diverse needs of users in mind, incorporating a flexible transaction fee mechanism. The default fee is set at 1% per transaction, but users are allowed to adjust the rate according to their needs. However, the fee is strictly capped at a maximum of 100% to prevent extreme scenarios.

Figure 1

SiuToken on the Sepolia testnet



Etherscan

Token SiuToken (SiuToken)

ERC-20

Overview

MAX TOTAL SUPPLY
1,000,000 SiuToken

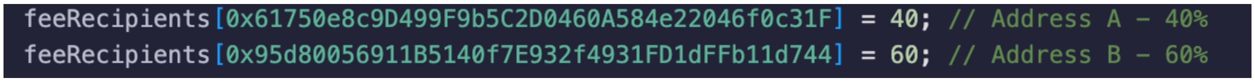
HOLDERS
11

TOTAL TRANSFERS
66

When a user initiates a token transfer, the contract automatically calculates the transaction fee based on the transfer amount and distributes it according to a preset ratio: 40% of the fee is sent to a designated address A, while the remaining 60% goes to address B (Figure 2). This fee mechanism is intended to incentivize various ecosystem participants to actively engage in the use and promotion of the token.

Figure 2

The Fee Mechanism



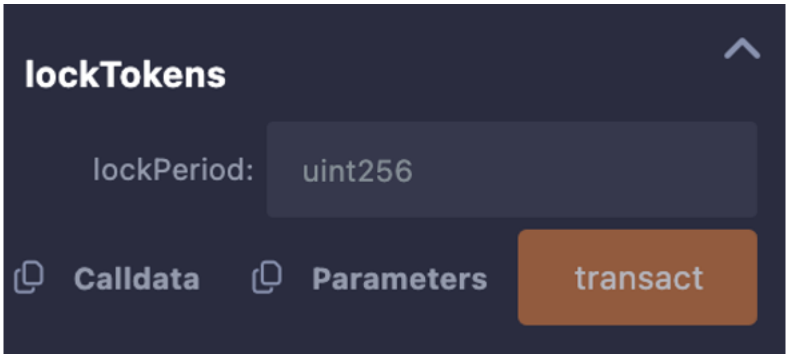
```
feeRecipients[0x61750e8c9D499F9b5C2D0460A584e22046f0c31F] = 40; // Address A - 40%
feeRecipients[0x95d80056911B5140f7E932f4931FD1dFFb11d744] = 60; // Address B - 60%
```

However, since the fee adjustment is currently open to all users, there is a potential risk of malicious intent or user error resulting in abnormally high fee settings. Future versions need to implement role-based permission controls to better regulate fee adjustment privileges.

In addition, the contract includes a token locking feature, allowing users to lock their tokens for a customizable period using the lockTokens function (Figure 3). During this lock period, all token transfers are prohibited, helping to manage the circulating supply of tokens in the market and prevent excessive price volatility. Each user's unlock time is recorded through an internal mapping structure and checked in the transferWithLockCheck function to ensure that tokens cannot be transferred during the lock period.

Figure 3

LockTokens function

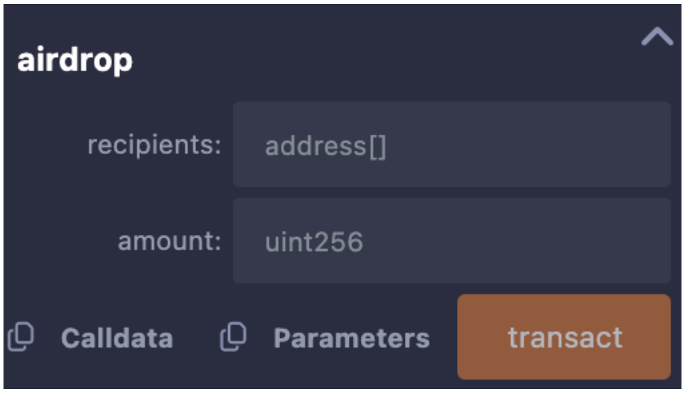


Although this feature regulates market circulation, there is currently no emergency unlock or early withdrawal mechanism. As a result, if a user mistakenly sets an excessively long lock period, their funds could be frozen and inaccessible for an extended time. It is recommended that future versions include special conditions or an administrator role to enable emergency unlocking when necessary.

The airdrop function is designed to support community building and token distribution (Figure 4). It allows users to distribute tokens to multiple recipients in a single transaction, enhancing the contract's utility and flexibility. However, the current airdrop mechanism lacks a pre-check for sufficient user balances. If a user attempts an airdrop without having enough tokens, the transaction fails or triggers a blockchain transaction rollback. It is recommended that future versions implement a balance verification step before executing airdrops.

Figure 4

Airdrop function

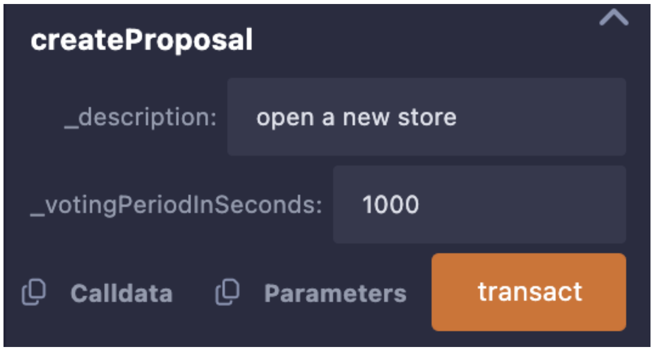


Technical Design and Implementation of the SiuDAO Contract

This project also developed a decentralized autonomous organization (DAO) contract, SiuDAO, built on top of the SiuToken. The contract primarily provides governance functions such as proposal creation, voting, proposal execution, and query access. Users can create new proposals using the createProposal function, which records the proposal's description, voting deadline, and the proposer’s address. Once created, the proposal is broadcast through the blockchain’s event system to notify the network (Figure 5). Figure 6 demonstrate the transaction was executed to create a new governance proposal on the SiuDAO smart contract. The proposal, titled "open a new store", was successfully submitted by the user 0x617...c31f and assigned Proposal ID 0. The voting period was set to 1000 seconds, and the proposal is now ready for community voting.

Figure 5

CreateProposal Function



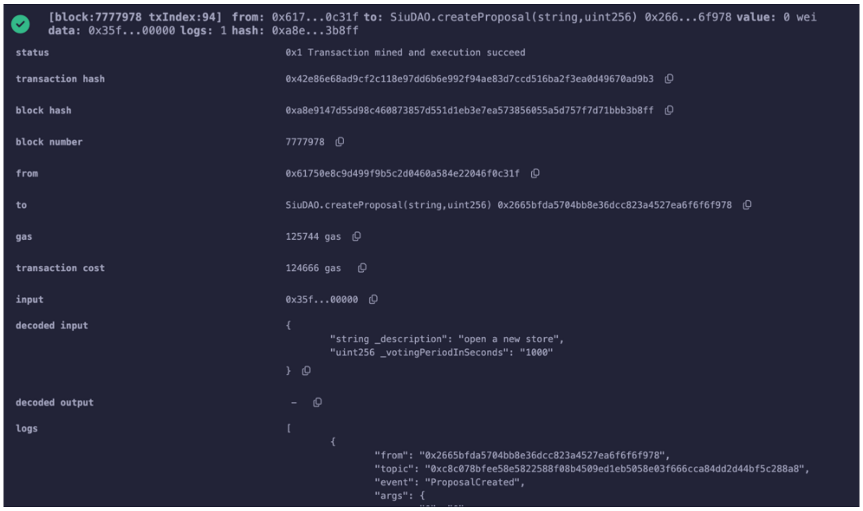
createProposal

_description: open a new store

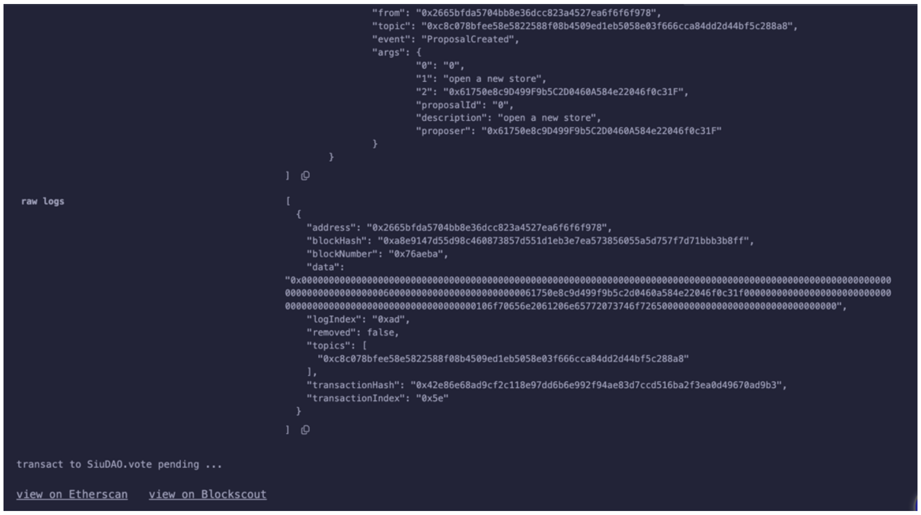
_votingPeriodInSeconds: 1000

Figure 6

SiuDAO Proposal #0 Created



```
[block:7777978 txIndex:94] from: 0x617...0c31f to: SiuDAO.createProposal(string,uint256) 0x266...6f978 value: 0 wei
data: 0x35f...00000 logs: 1 hash: 0xa8e...3b8ff
status                          0x1 Transaction mined and execution succeed

transaction hash                0x42e86e68ad9cf2c118e97dd6b6e992f94ae83d7ccd516ba2f3ea0d49670ad9b3

block hash                      0xa8e9147d55d98c460873857d551d1eb3e7ea573856055a5d757f7d71bbb3b8ff

block number                    7777978

from                            0x61750e8c9d499f9b5c2d0460a584e22046f0c31f

to                              SiuDAO.createProposal(string,uint256) 0x2665bfda5704bb8e36dcc823a4527ea6f6f6f978

gas                             125744 gas

transaction cost                124666 gas

input                           0x35f...00000

decoded input                   {
                                        "string _description": "open a new store",
                                        "uint256 _votingPeriodInSeconds": "1000"
                                } 

decoded output                   - 

logs                            [
                                        {
                                                "from": "0x2665bfda5704bb8e36dcc823a4527ea6f6f6f978",
                                                "topic": "0xc8c078bfee58e5822588f08b4509ed1eb5058e03f666cca84dd2d44bf5c288a8",
                                                "event": "ProposalCreated",
                                                "args": {
```

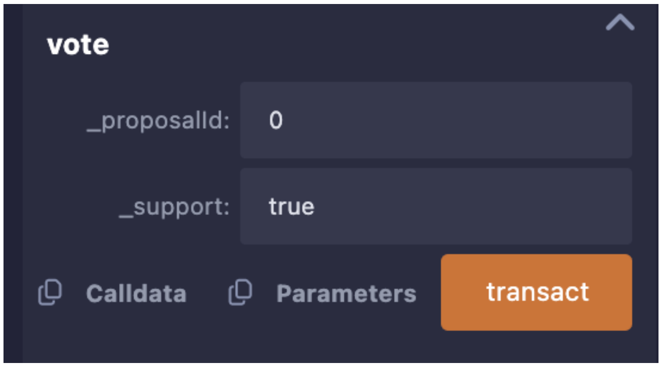


```
                                                "from": "0x2665bfda5704bb8e36dcc823a4527ea6f6f6f978",
                                                "topic": "0xc8c078bfee58e5822588f08b4509ed1eb5058e03f666cca84dd2d44bf5c288a8",
                                                "event": "ProposalCreated",
                                                "args": {
                                                        "0": "0",
                                                        "1": "open a new store",
                                                        "2": "0x61750e8c9D499F9b5C2D0460A584e22046f0c31F",
                                                        "proposalId": "0",
                                                        "description": "open a new store",
                                                        "proposer": "0x61750e8c9D499F9b5C2D0460A584e22046f0c31F"
                                                }
                                        }
                                ] 

 raw logs                       [
                                  {
                                    "address": "0x2665bfda5704bb8e36dcc823a4527ea6f6f6f978",
                                    "blockHash": "0xa8e9147d55d98c460873857d551d1eb3e7ea573856055a5d757f7d71bbb3b8ff",
                                    "blockNumber": "0x76aeba",
                                    "data":
                                "0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                                00000000000000000006000000000000000000000000061750e8c9d499f9b5c2d0460a584e22046f0c31f0000000000000000000000000
                                0000000000000000000000000000000000000106f70656e2061206e65772073746f726500000000000000000000000000000000",
                                    "logIndex": "0xad",
                                    "removed": false,
                                    "topics": [
                                      "0xc8c078bfee58e5822588f08b4509ed1eb5058e03f666cca84dd2d44bf5c288a8"
                                    ],
                                    "transactionHash": "0x42e86e68ad9cf2c118e97dd6b6e992f94ae83d7ccd516ba2f3ea0d49670ad9b3",
                                    "transactionIndex": "0x5e"
                                  }
                                ] 

transact to SiuDAO.vote pending ...

view on Etherscan    view on Blockscout
```

The voting function allows users to cast votes in support or opposition based on the number of SiuToken tokens they hold (Figure 7). Each address is allowed to vote only once per proposal during the voting period. The contract enforces this rule through internal mappings that strictly check for duplicate voting to ensure fairness.

Figure 7

Voting Function



vote

_proposalId: 0

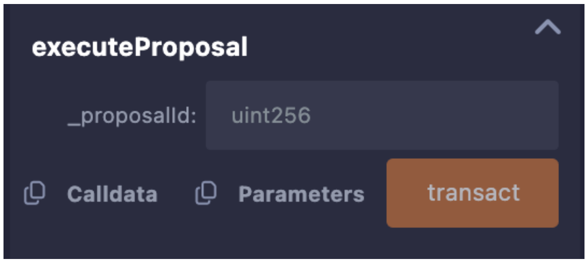
_support: true

However, the current vote weighting is based solely on the user's real-time token balance at the time of voting, rather than using a snapshot mechanism. This could allow users to manipulate the outcome by rapidly transferring tokens in and out within the voting window. To improve fairness, it is recommended that future versions implement a snapshot-based or token-locking voting system.

Regarding proposal execution, once the voting period ends, users can mark a proposal as completed using the executeProposal function (Figure 8). In its current implementation, the contract only provides a status update feature and does not yet include actual execution logic or interactions with other smart contracts. Future versions should expand this functionality to allow proposals to trigger real actions.

Figure 8

ExecuteProposal Function



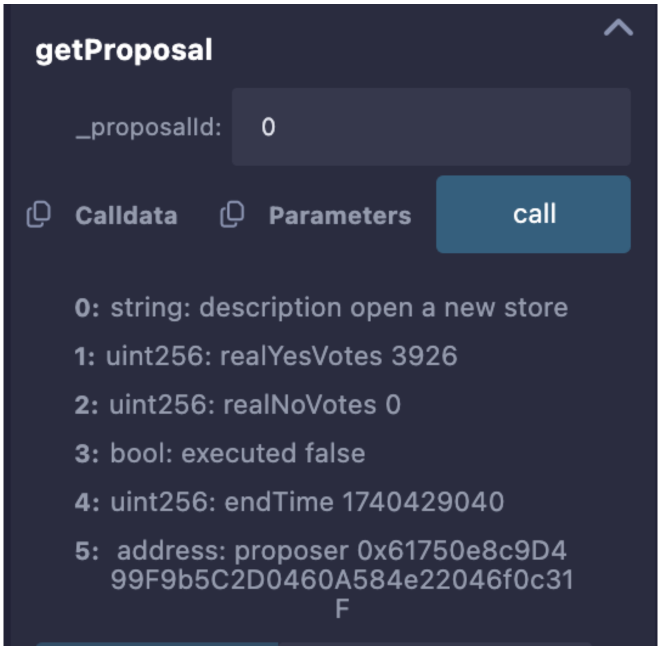
executeProposal

_proposalId: uint256

Additionally, to enhance the user experience in governance participation, the contract includes a getProposal function that allows users to view voting results in a readable format (Figure 9). However, this function currently lacks real-time updates, meaning users cannot monitor voting progress dynamically. It is recommended that future versions incorporate live tracking features to improve the transparency of the governance process.

Figure 9

GetProposal Function



getProposal

_proposalId: 0

```
0: string: description open a new store
1: uint256: realYesVotes 3926
2: uint256: realNoVotes 0
3: bool: executed false
4: uint256: endTime 1740429040
5:  address: proposer 0x61750e8c9D4
    99F9b5C2D0460A584e22046f0c31
                 F
```

By thoroughly analyzing the technical design of smart contracts and identifying potential risks, I will be well-equipped to contribute to future upgrades if I work in a related industry. This capability will allow me to enhance contract security and usability, ultimately supporting the sustainable and long-term development of the blockchain ecosystem.

Although blockchain technology may not appear directly related to hedge funds or asset management, it is increasingly becoming an area of strategic interest. If a hedge fund or asset management firm were to explore blockchain in the future, I would be a strong candidate to lead such initiatives. Unlike most university students, I already possess hands-on experience with smart contracts and decentralized systems—skills that are still relatively rare in the industry. This enables me to bring more than just financial expertise to a project; I can also contribute substantial blockchain development knowledge, making me a well-rounded and versatile talent.