SafeClub – Trésorerie sécurisée d'un club étudiant sur Ethereum

1. But du projet

SafeClub est un contrat intelligent (smart contract) qui permet à un club étudiant de gérer sa **trésorerie en ETH** (de l'argent sur la blockchain Ethereum), de façon transparente et sécurisée.

Fonctionnalités principales :

- **Gestion des membres** : le club peut ajouter ou retirer des membres qui ont le droit de participer.
- **Réception d'ETH** : le contrat peut recevoir des fonds (argent) en ETH, comme une "cagnotte".
- **Création de propositions de dépenses** : les membres peuvent proposer des dépenses avec un montant, un bénéficiaire, une description, et un délai pour voter.
- **Vote des membres** : chaque membre peut voter pour ou contre chaque proposition (1 seul vote par membre par proposition).
- **Exécution sécurisée** : si une proposition est acceptée (majorité + quorum), l'argent est transféré au bénéficiaire, mais seulement si toutes les règles sont respectées (vérifications de sécurité).

Explication code :

Compile | Home | safeclub.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import "https://cdn.jsdelivr.net/npm/@openzeppelin/contracts@4.9.3/access/Ownable.sol";
import "https://cdn.jsdelivr.net/npm/@openzeppelin/contracts@4.9.3/security/ReentrancyGuard.sol";

contract SafeClub is Ownable, ReentrancyGuard {
    struct Proposal {
        uint256 id;
        address payable to;
        uint256 amount;
        string description;
        uint256 deadline;
        uint256 votesFor;
        uint256 votesAgainst;
        bool executed;
        uint256 createdAt;
    }
    mapping(address => bool) public members;
    uint256 public memberCount;

    mapping(uint256 => Proposal) public proposals;
    uint256 public proposalCount;
```

Compile | Home | safeclub.sol

```
    mapping(uint256 => mapping(address => bool)) public hasVoted;

    uint8 public quorumPercent = 50;

    event MemberAdded(address indexed member);
    event MemberRemoved(address indexed member);
    event Deposit(address indexed from, uint256 amount);
    event ProposalCreated(uint256 id, address indexed proposer, address to, uint256 amount, uint256 deadline);
    event Voted(uint256 indexed id, address indexed voter, bool support);
    event ProposalExecuted(uint256 id, address indexed executor);

    modifier onlyMember() {
        require(members[msg.sender], "Only members");
        _;
    }

    constructor() {    infinite gas 1936400 gas
        members[msg.sender] = true;
        memberCount = 1;
        emit MemberAdded(msg.sender);
    }
```

Compile | Home | safeclub.sol

```
    receive() external payable {    undefined gas
        emit Deposit(msg.sender, msg.value);
    }

    function deposit() external payable {    1733 gas
        emit Deposit(msg.sender, msg.value);
    }

    function addMember(address _member) external onlyOwner {    infinite gas
        require(_member != address(0), "zero address");
        require(!members[_member], "already member");
        members[_member] = true;
        memberCount += 1;
        emit MemberAdded(_member);
    }

    function removeMember(address _member) external onlyOwner {    infinite gas
        require(members[_member], "not a member");
        members[_member] = false;
        memberCount -= 1;
        emit MemberRemoved(_member);
    }
```

Compile | Home | safeclub.sol

```
    function createProposal(    undefined gas
        address payable _to,
        uint256 _amountWei,
        string calldata _description,
        uint256 _durationSeconds
    ) external onlyMember returns (uint256) {
        require(_to != address(0), "invalid recipient");
        require(_amountWei > 0, "amount > 0");
        require(_durationSeconds >= 60, "duration too short");

        proposalCount += 1;
        uint256 id = proposalCount;

        proposals[id] = Proposal({
            id: id,
            to: _to,
            amount: _amountWei,
            description: _description,
            deadline: block.timestamp + _durationSeconds,
            votesFor: 0,
            votesAgainst: 0,
            executed: false,
            createdAt: block.timestamp
```

Compile | Home | safeclub.sol

```
            createdAt: block.timestamp
        });

        emit ProposalCreated(id, msg.sender, _to, _amountWei, proposals[id].deadline);
        return id;
    }

    function vote(uint256 _proposalId, bool _support) external onlyMember {    infinite gas
        Proposal storage p = proposals[_proposalId];
        require(p.id != 0, "proposal not found");
        require(block.timestamp <= p.deadline, "voting closed");
        require(!hasVoted[_proposalId][msg.sender], "already voted");
        require(!p.executed, "already executed");

        hasVoted[_proposalId][msg.sender] = true;

        if (_support) {
            p.votesFor += 1;
        } else {
            p.votesAgainst += 1;
        }

        emit Voted(_proposalId, msg.sender, _support);
```

```
Compile   Home   safeclub.sol
113         emit Voted(_proposalId, msg.sender, _support);
114     }
115
116     function isAccepted(uint256 _proposalId) public view returns (bool) {    infinite gas
117     Proposal storage p = proposals[_proposalId];
118     uint256 totalVotes = p.votesFor + p.votesAgainst;
119
120     uint256 minVotesRequired = 1;
121
122     if (totalVotes < minVotesRequired) return false;
123
124     return p.votesFor > p.votesAgainst;
125 }
126
127     function executeProposal(uint256 _proposalId) exter    infinite gas
128         Proposal storage p = proposals[_proposalId];
129         require(block.timestamp > p.deadline, "voting not finished");
130         require(!p.executed, "already executed");
131         require(isAccepted(_proposalId), "proposal not accepted");
132         require(address(this).balance >= p.amount, "insufficient balance");
133
134         p.executed = true;
```

Literal

contracts/safeclub.sol 128:46

```
Compile   Home   safeclub.sol
131         require(isAccepted(_proposalId), "proposal not accepted");
132         require(address(this).balance >= p.amount, "insufficient balance");
133
134         p.executed = true;
135
136         (bool sent, ) = p.to.call{value: p.amount}("");
137         require(sent, "transfer failed");
138
139         emit ProposalExecuted(_proposalId, msg.sender);
140     }
141
142     function getContractBalance() external view returns (uint256) {    423 gas
143         return address(this).balance;
144     }
145 }
146
```

a) Import OpenZeppelin

import "https://cdn.jsdelivr.net/npm/@openzeppelin/contracts@4.9.3/access/Ownable.sol";

import "https://cdn.jsdelivr.net/npm/@openzeppelin/contracts@4.9.3/security/ReentrancyGuard.sol";

-**Ownable** : fournit une fonctionalité de propriétaire unique (owner) du contrat qui peut faire certaines opérations exclusives

- **ReentrancyGuard** : protège contre les attaques de type "reentrancy" où un attaquant pourrait appeler plusieurs fois une fonction de transfert d'argent pour voler des fonds.

b) Structure Proposal

```
struct Proposal {
   uint256 id;
   address payable to;
   uint256 amount;
   string description;
   uint256 deadline;
   uint256 votesFor;
   uint256 votesAgainst;
   bool executed;
   uint256 createdAt;
}
```

c) Gestion des membres

```
mapping(address => bool) public members;
uint256 public memberCount;
```

- members : mapping (adresse → bool) indiquant qui est membre (true) ou non (false)
- memberCount : nombre total de membres actifs

Le **propriétaire** (owner) peut ajouter ou retirer des membres avec ces fonctions :

```
function addMember(address _member) external onlyOwner { ... }
function removeMember(address _member) external onlyOwner { ... }
```

d) Réception d'ETH

```
receive() external payable { ... }
function deposit() external payable { ... }
```

- Le contrat peut recevoir des ETH via la fonction spéciale receive() (transfert direct) ou via la fonction deposit() explicite.

-Un événement est émis pour enregistrer chaque dépôt.

e) Création des propositions

```
function createProposal(
   address payable _to,
   uint256 _amountWei,
   string calldata _description,
   uint256 _durationSeconds
) external onlyMember returns (uint256) { ... }
```

- Un membre peut proposer une dépense : montant, destinataire, description et durée de vote (minimum 60 secondes).

- La proposition est enregistrée, un ID est attribué, et un événement est émis.

f) Vote des membres

```
function vote(uint256 _proposalId, bool _support) external onlyMember { ... }
```

- Un membre peut voter **une seule fois** par proposition, pour ou contre.

- Le vote est enregistré et un événement est émis.

- Le vote n'est possible que si la date limite n'est pas dépassée et que la proposition n'est pas déjà exécutée.

g) Vérification d'acceptation

```
function isAccepted(uint256 _proposalId) public view returns (bool) { ... }
```

- Pour qu'une proposition soit acceptée :
    - Il faut atteindre un **quorum** (exemple : au moins 50% des membres ont voté)
    - Il faut que les votes pour soient plus nombreux que les votes contre

h) Exécution sécurisée

```
function executeProposal(uint256 _proposalId) external nonReentrant onlyMember { ... }
```

Un membre peut exécuter une proposition uniquement si :

- La période de vote est terminée
- La proposition est acceptée
- Elle n'a pas encore été exécutée
- Le contrat a assez d'ETH pour payer

La fonction est protégée contre la reentrancy (nonReentrant)

Le montant est envoyé au bénéficiaire

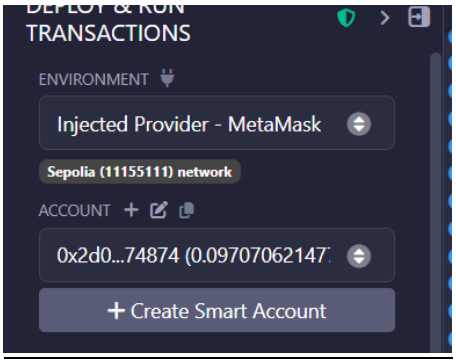
Un événement est émis

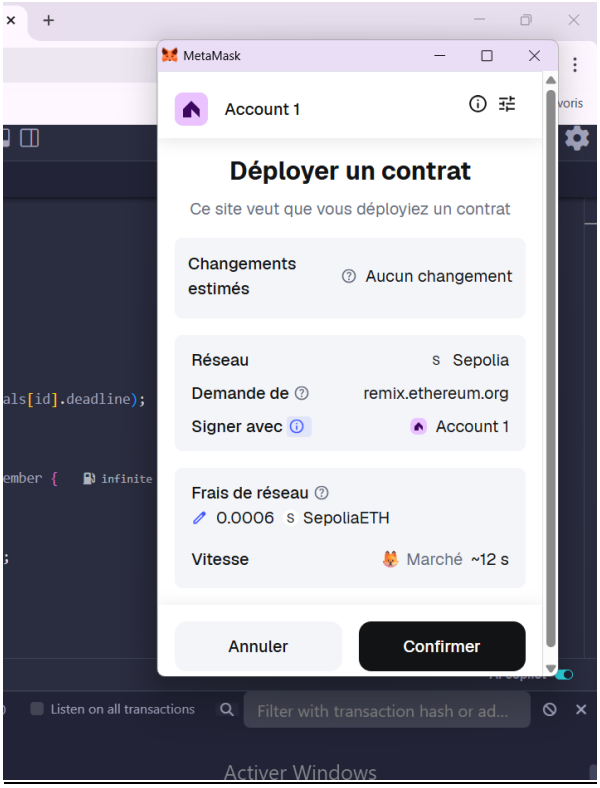
i) Fonction utilitaire

```
function getContractBalance() external view returns (uint256) {
   return address(this).balance;
}
```

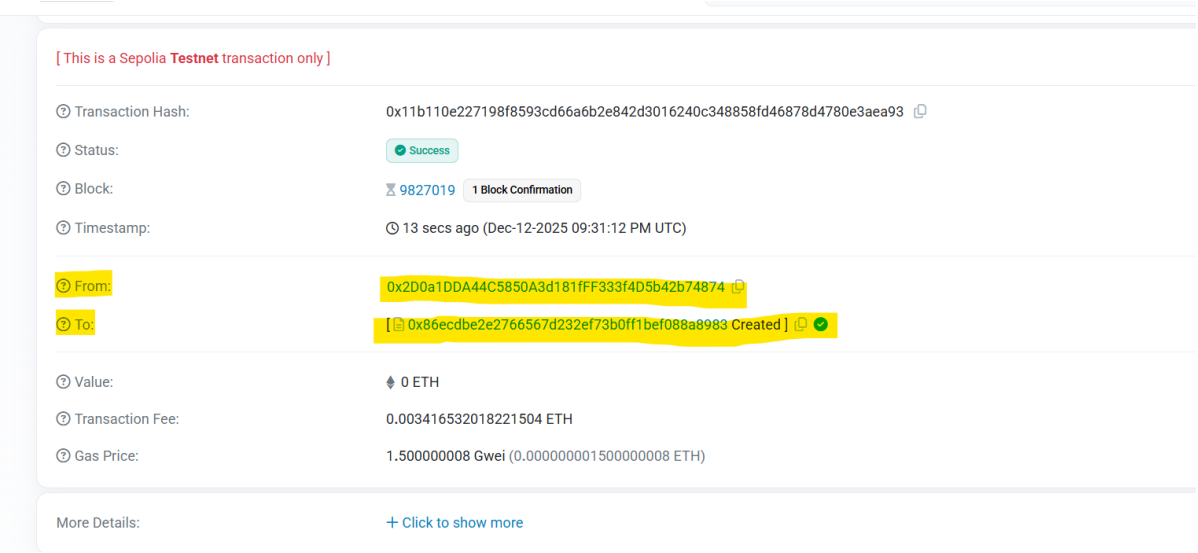
Permet de voir combien d'ETH est stocké dans le contrat.

Deployment





Etherscan spécifique au réseau de test Sepolia



Détail de ta transaction sur cette page :

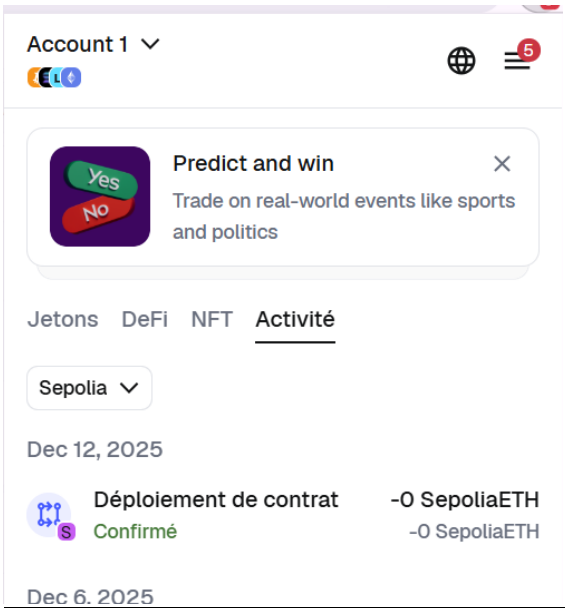
- Transaction Hash :
  C'est un identifiant unique qui représente ta transaction sur la blockchain.
  Ici :
  0x11b110e227198f8593cd66a6b2e842d3016240c348858fd46878d4780e3aea93

- Status: Success :
  La transaction a été validée avec succès, donc ton contrat a bien été déployé.

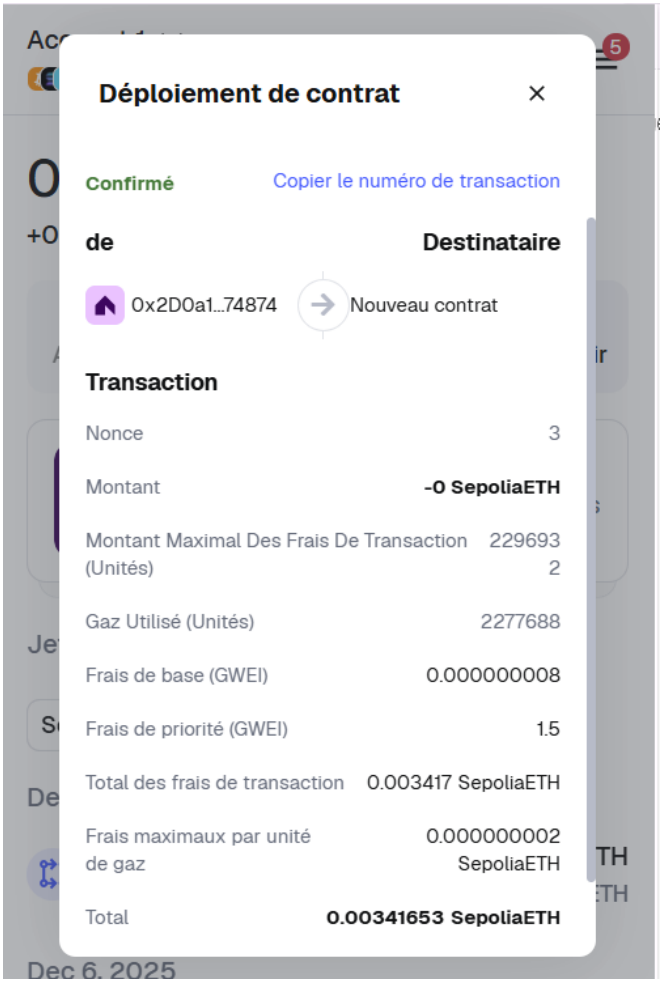
- Block : 9827019 :
  La transaction est incluse dans ce bloc de la blockchain.

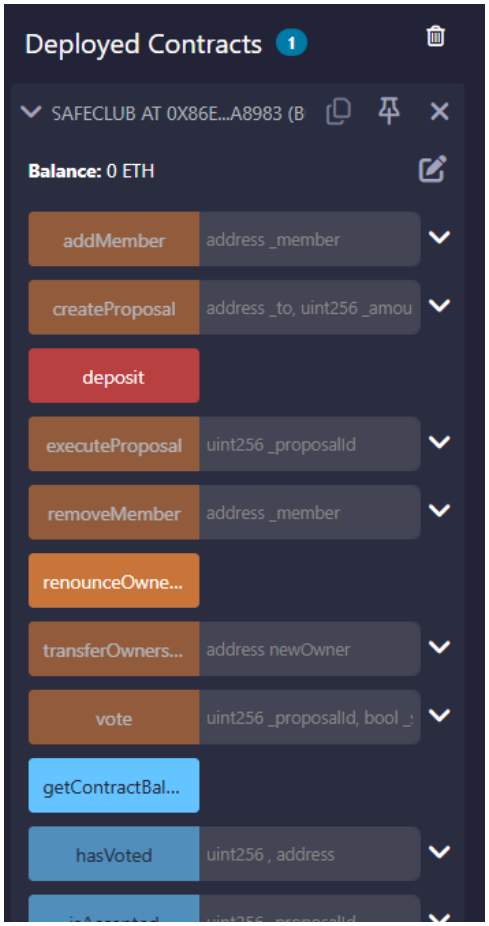
- **Timestamp** :
  L'heure précise à laquelle la transaction a été validée (ici, il y a 13 secondes).
- **From** :
  L'adresse Ethereum qui a envoyé la transaction (ton compte MetaMask, celui qui a déployé le contrat).
- **To** :
  L'adresse du contrat déployé (le résultat de ta transaction).
  C'est l'adresse où ton smart contract SafeClub est désormais disponible sur Sepolia.
- **Value** :
  Montant envoyé avec la transaction, ici 0 ETH car tu déployais juste un contrat.
- **Transaction Fee** :
  Frais de transaction payés (en ETH) pour faire cette opération.
- **Gas Price** :
  Le prix du gas au moment de la transaction (combien tu payes par unité de gas).

l'adresse du contrat : 0xdc005b33Fe8b5608E6Bf23498Bc98447997ECa9A

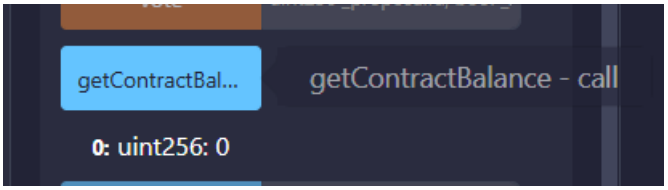


=>contrat bien deployé





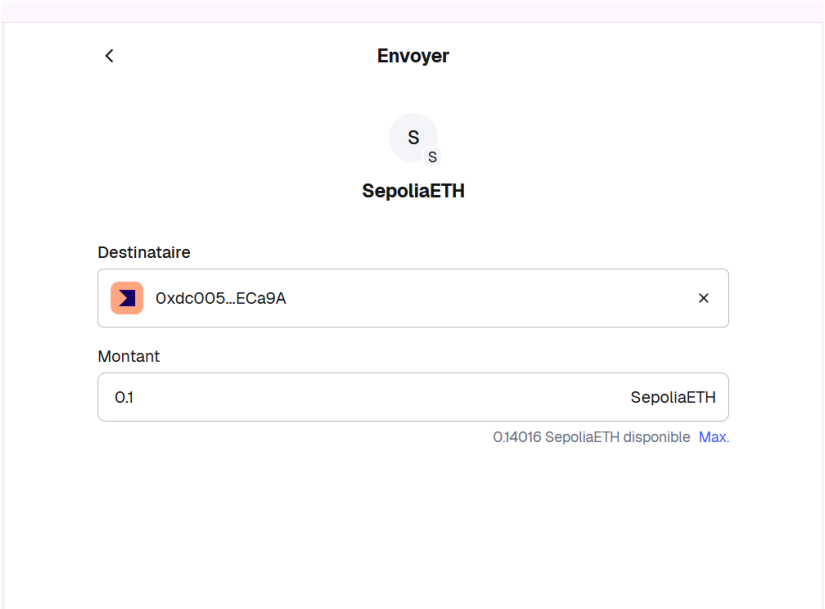
1.



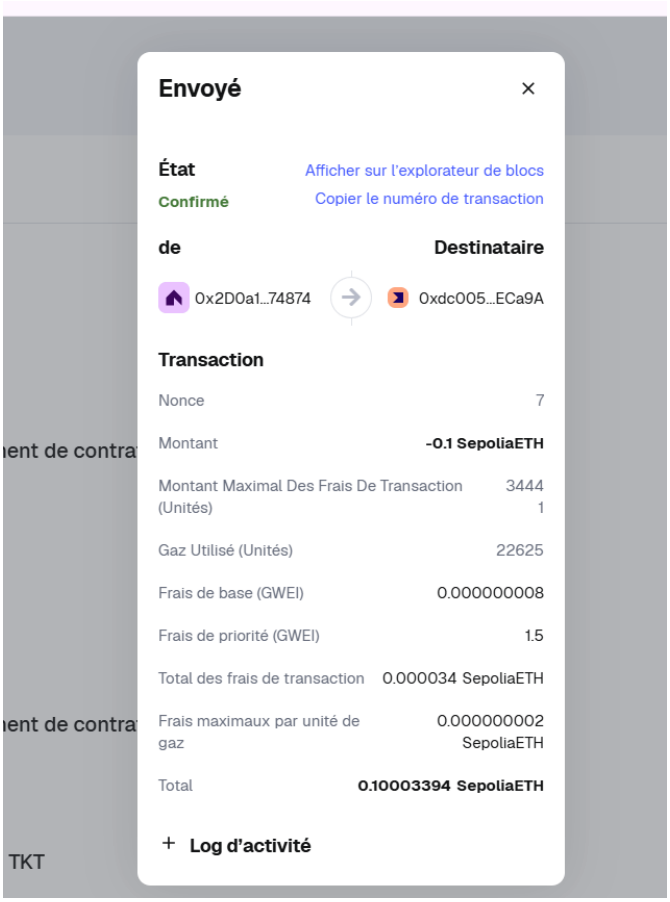
n’as pas encore envoyé d’ETH

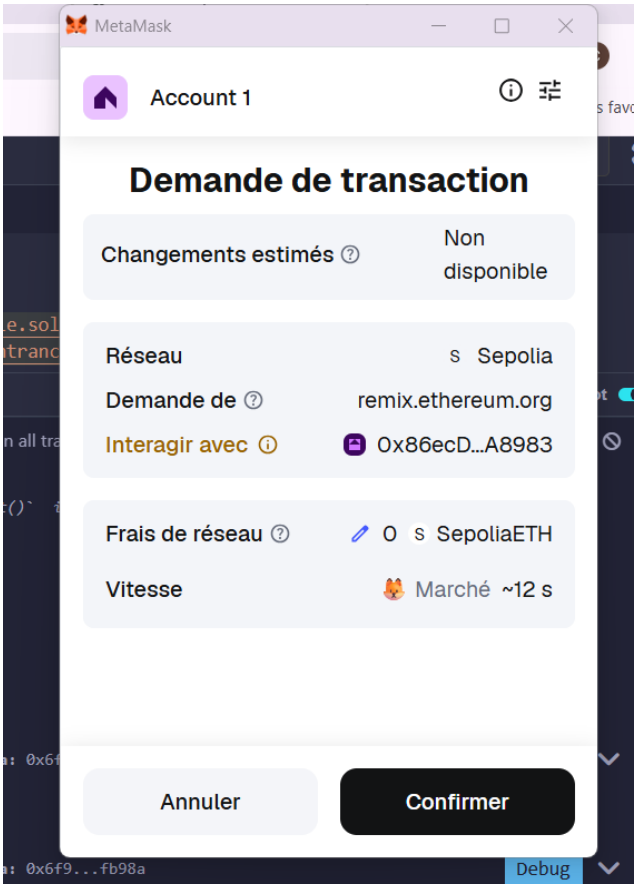
2.

Envoyer de l’ETH au contrat (alimenter la trésorerie)



=> envoie une petite somme (exemple 0.1 ETH) depuis metamask a l adresse de contrat


Envoyé
État
Confirmé
Afficher sur l'explorateur de blocs
Copier le numéro de transaction
de
Destinataire
0x2D0a1...74874
0xdc005...ECa9A
Transaction
Nonce 7
Montant -0.1 SepoliaETH
Montant Maximal Des Frais De Transaction (Unités) 3444 1
Gaz Utilisé (Unités) 22625
Frais de base (GWEI) 0.000000008
Frais de priorité (GWEI) 1.5
Total des frais de transaction 0.000034 SepoliaETH
Frais maximaux par unité de gaz 0.000000002 SepoliaETH
Total 0.10003394 SepoliaETH
Log d'activité
TKT


MetaMask
Account 1
Demande de transaction
Changements estimés
Non disponible
Réseau
Sepolia
Demande de
remix.ethereum.org
Interagir avec
0x86ecD...A8983
Frais de réseau
0 SepoliaETH
Vitesse
Marché ~12 s
Annuler
Confirmer
Debug

```
CALL  [call] from: 0x2d0a1dda44c5850a3d181fff333f4d5b42b74874 to: SafeClub.getContractBalance() data: 0x6f9...fb98a

from            0x2D0a1DDA44C5850A3d181fFF333f4D5b42b74874
to              SafeClub.getContractBalance() 0xdc005b33Fe8b5608E6Bf23498Bc98447997ECa9A
input           0x6f9...fb98a
output          0000000000000000000000000199691209313800
decoded input   {}
decoded output  {
                  "0": "uint256: 100000000000000000"
                }
logs            []
raw logs        []
```

3. creation membre

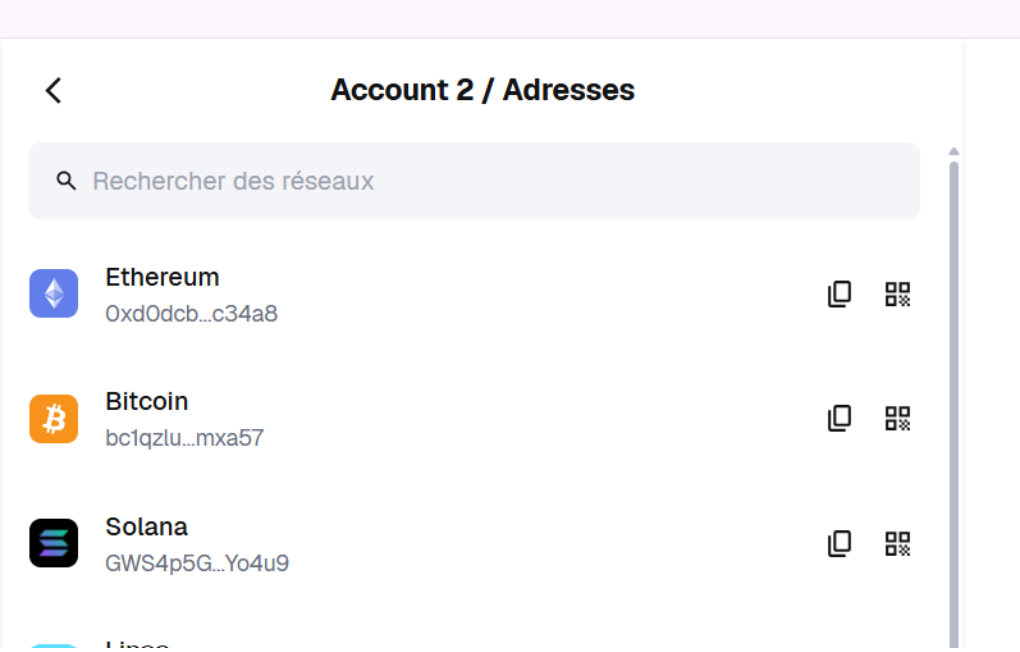
Owner

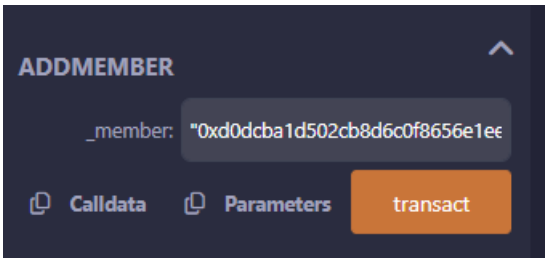
owner

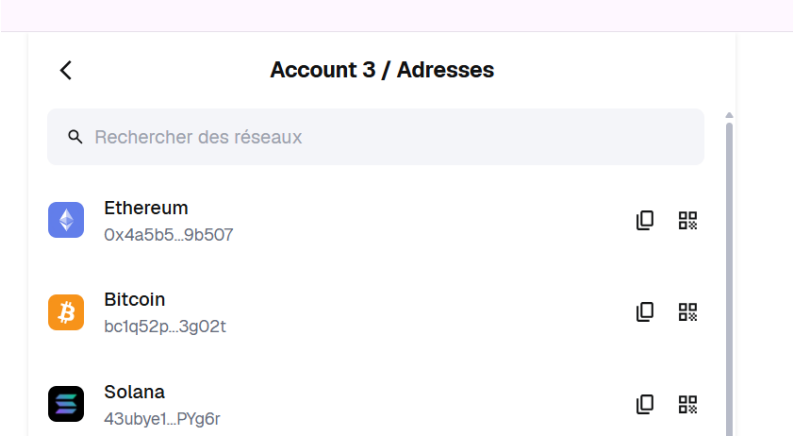
owner - call

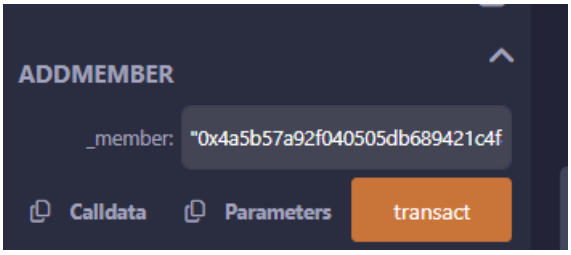
0: address: 0x2D0a1DDA44C5850A3d181fFF333f4D5b42b74874

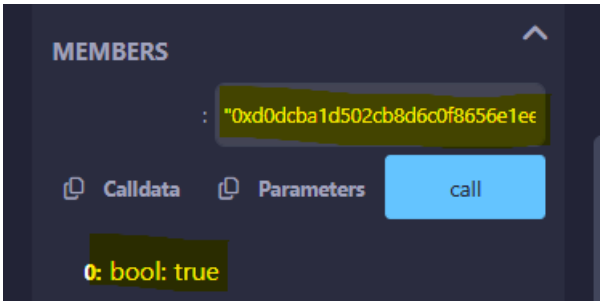
Sur metamask creation d un 2eme account



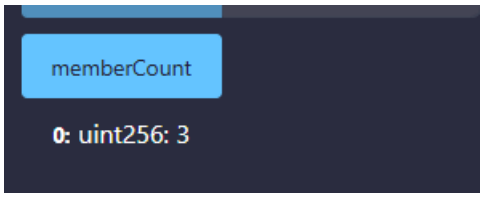






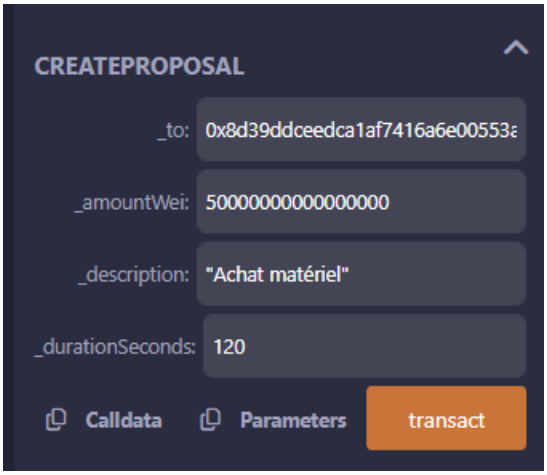


⇨ Les membres ont été bien ajouté members return true



⇨ 2membres

4. **créer une proposition de dépense**



la **période de vote= 2mins**

ID de proposition= 2

2 proposals

proposalCount

0: uint256: 2

PROPOSALS

: 1

Calldata Parameters call

0: uint256: id 1

1: address: to 0x8d39ddCeEDca1Af7416A6e00553A37Bd52D5B17a

2: uint256: amount 50000000000000000

3: string: description Achat matériel

4: uint256: deadline 1765579056

5: uint256: votesFor 0

6: uint256: votesAgainst 0

7: bool: executed false

8: uint256: createdAt 1765578936

PROPOSALS

: 2

Calldata Parameters call

0: uint256: id 2

1: address: to 0x8d39ddCeEDca1Af7416A6e00553A37Bd52D5B17a

2: uint256: amount 50000000000000000

3: string: description Achatl

4: uint256: deadline 1765579804

5: uint256: votesFor 1

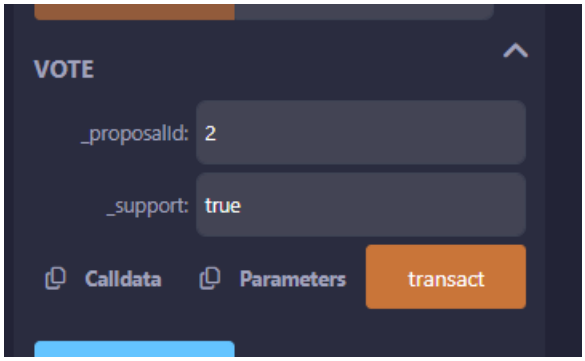
6: uint256: votesAgainst 0

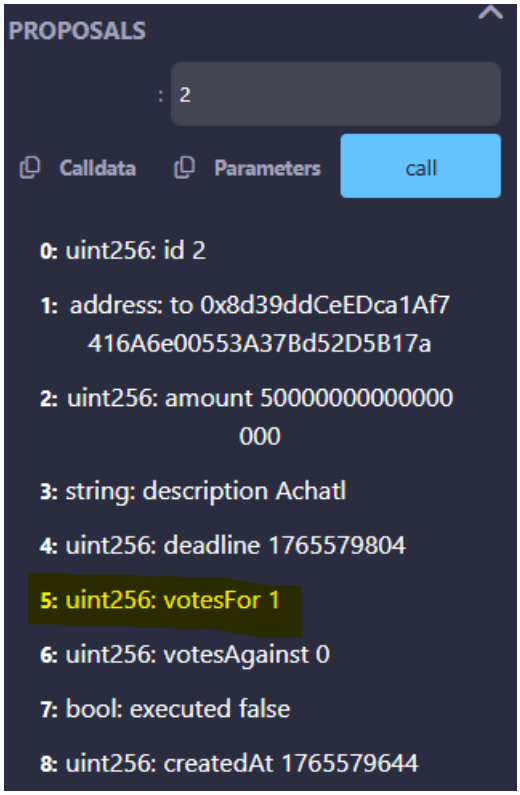
7: bool: executed false

8: uint256: createdAt 1765579644

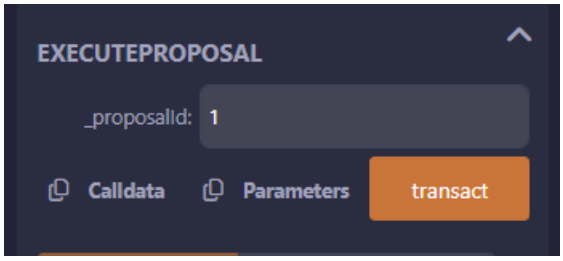
5.Le vote

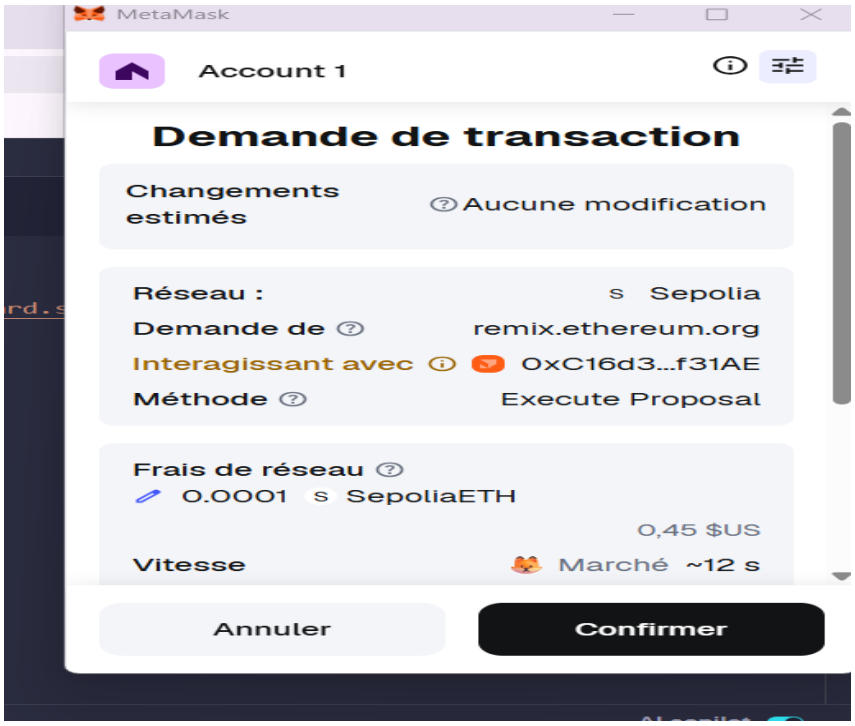
Changer le compte dans metamask a chaque fois pour voter



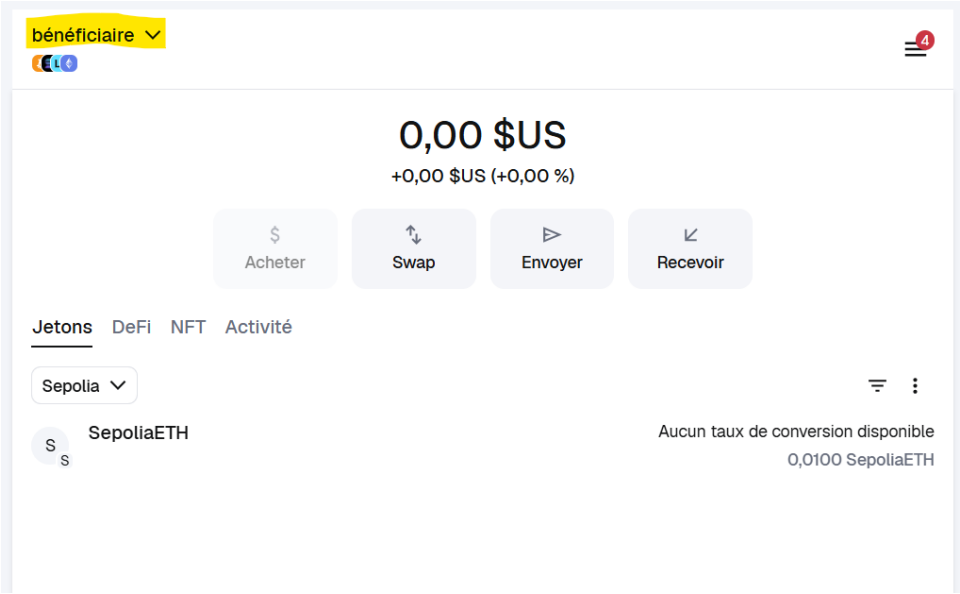


6.Execution de Proposal

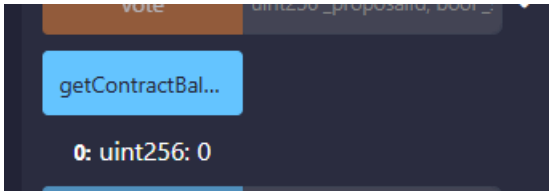



MetaMask
Account 1
Demande de transaction
Changements estimés
Aucune modification
Réseau :
Sepolia
Demande de
remix.ethereum.org
Interagissant avec
0xC16d3...f31AE
Méthode
Execute Proposal
Frais de réseau
0.0001 SepoliaETH
0,45 $US
Vitesse
Marché ~12 s
Annuler
Confirmer

Execute Proposal
Statut
Confirmé
Afficher sur l'explorateur de blocs
Copier le numéro de transaction
De la part de
Vers
0x2D0a1...74874
0xC16d3...f31AE
Transaction
Nonce
38
Montant
-0 SepoliaETH
Limite De Gaz (Unités)
101281
Gaz Utilisé (Unités)
96251
Frais de base (GWEI)
0.00000001
Frais de priorité (GWEI)
1.5
Montant total des frais de gaz
0.000144 SepoliaETH
Frais maximaux par unité de gaz
0.000000002 SepoliaETH
Total
0.00014438 SepoliaETH
Log d'activité



=>l'adresse bénéficiaire a bien **reçu l'ETH**



=>le solde du contrat **diminue (devient 0)**

Modèle de menaces

Scénario 1 : Attaque par réentrance (Reentrancy Attack)

Un attaquant pourrait exploiter une fonction effectuant un appel externe (call) avant de mettre à jour l'état interne, provoquant un double retrait ou une manipulation des fonds.
Impact : Perte de fonds du contrat.
Contre-mesure : Utilisation d'un modificateur nonReentrant empêchant les appels réentrants.

Scénario 2 : Manipulation du temps (Timestamp Manipulation)

Les mineurs peuvent ajuster légèrement block.timestamp, influençant les délais de vote pour des propositions.
Impact : Un mineur pourrait précipiter ou retarder la clôture des votes.
Contre-mesure : Le timestamp est uniquement utilisé pour gérer des délais peu critiques et ne sert pas à des calculs financiers sensibles.

Scénario 3 : Votes multiples ou non autorisés

Un membre pourrait tenter de voter plusieurs fois ou un tiers non membre pourrait essayer de voter.
Impact : Vote faussé et perte de confiance dans la gouvernance.
Contre-mesure : Contrôle strict des adresses autorisées et enregistrement des votes pour empêcher les doubles votes.

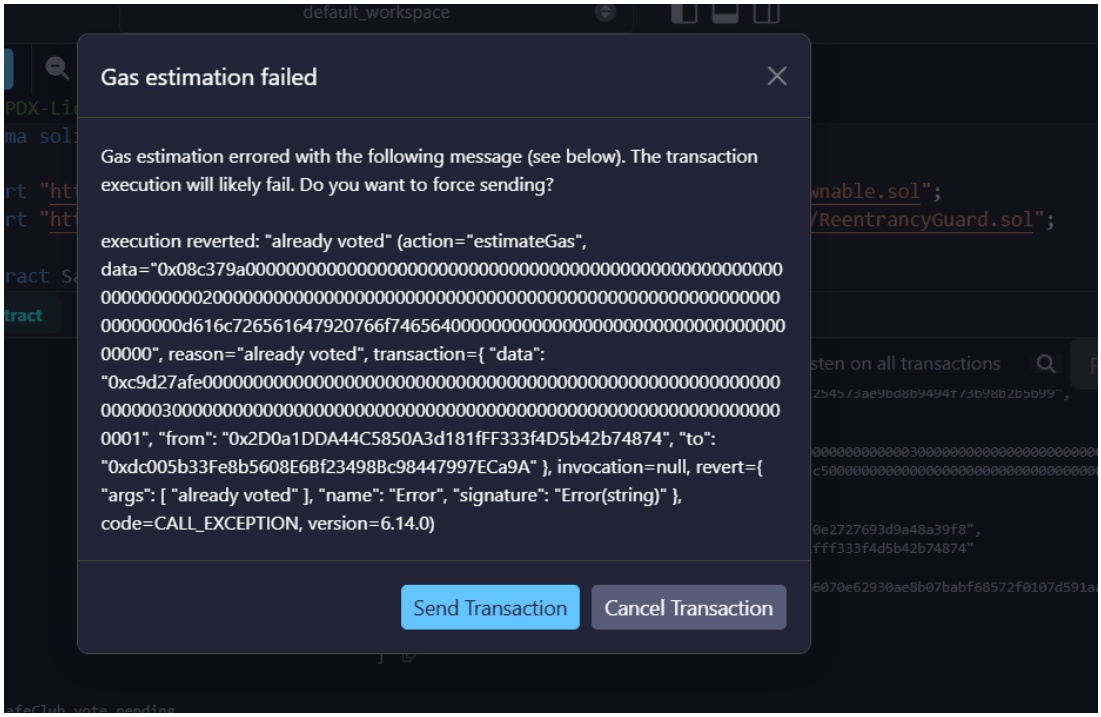
Scénario 4 : Dépassement de gaz (Gas Limit)

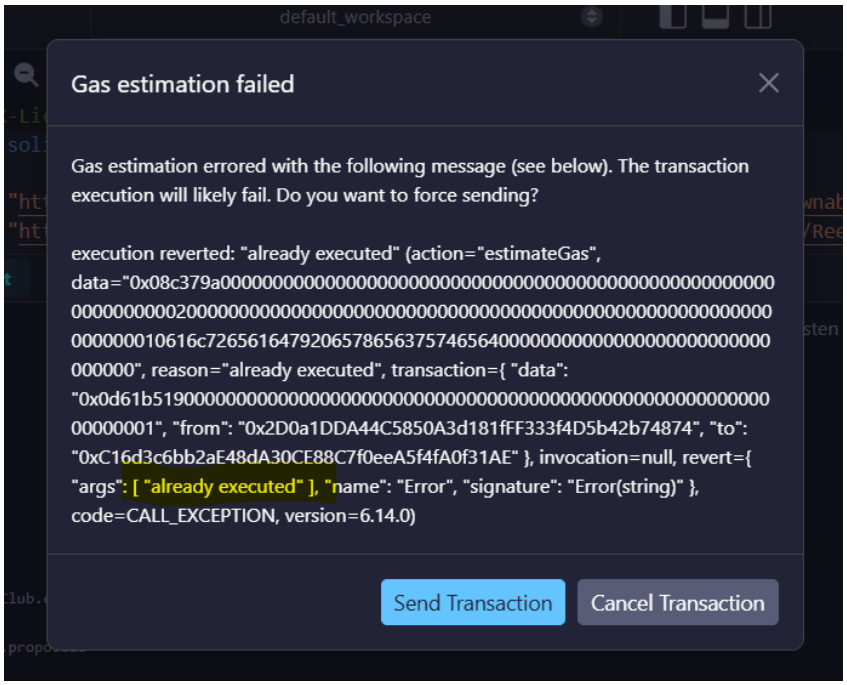
Fonctions impliquant des boucles sur de grandes structures de données pourraient dépasser la limite de gaz autorisée.
Impact : Blocage des fonctions critiques.
Contre-mesure : Conception évitant les boucles coûteuses, traitement optimisé des données.

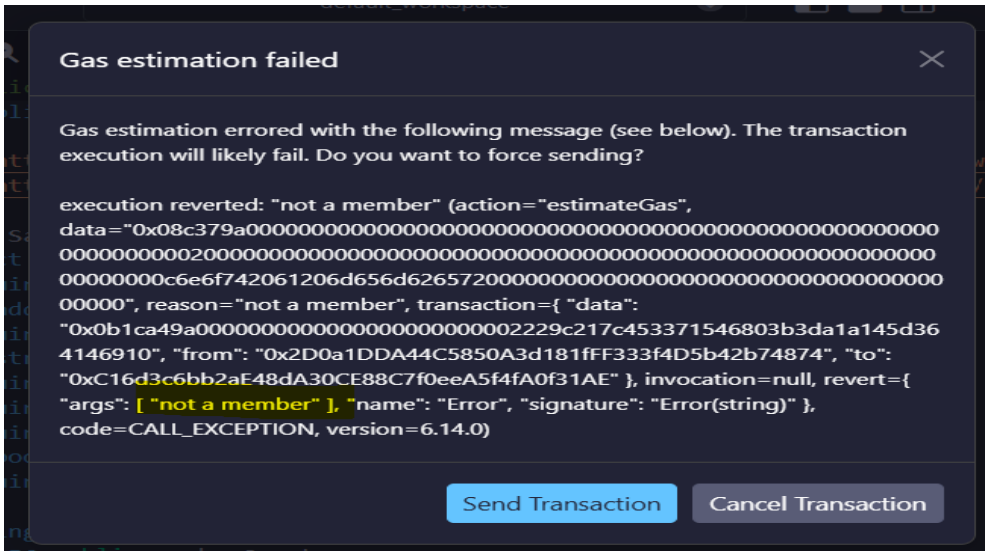
contre-mesures

=>Pas de double vote

=>Cela garantit l'intégrité des votes et empêche les votes non autorisés.

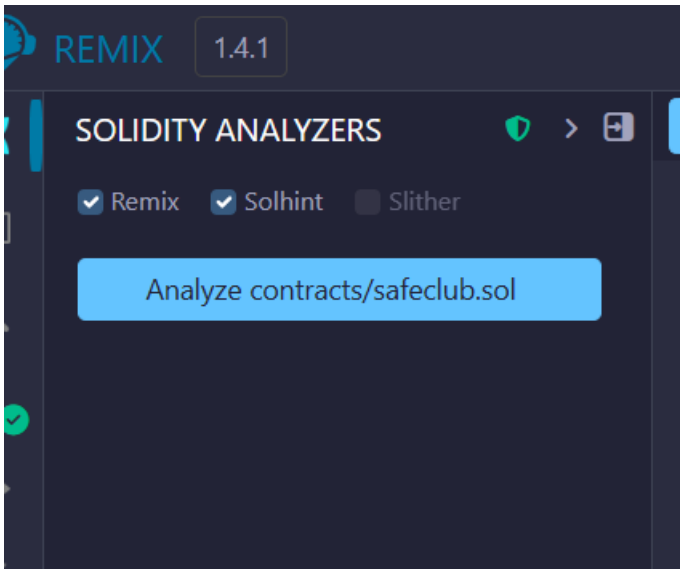

default_workspace
Gas estimation failed
Gas estimation errored with the following message (see below). The transaction execution will likely fail. Do you want to force sending?
execution reverted: "already voted" (action="estimateGas", data="0x08c379a0000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000


default_workspace
Gas estimation failed
Gas estimation errored with the following message (see below). The transaction execution will likely fail. Do you want to force sending?
execution reverted: "already executed" (action="estimateGas", data="0x08c379a0000000000000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000

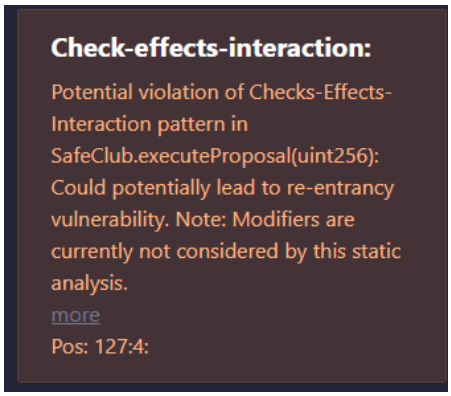

Gas estimation failed
Gas estimation errored with the following message (see below). The transaction execution will likely fail. Do you want to force sending?
execution reverted: "not a member" (action="estimateGas", data="0x08c379a0000000000

Analyse statique (détection des failles de sécurité, mauvaises pratiques ou risques logiques)

Outils : Remix Analyzer



a-



La fonction executeProposal effectue un appel externe (p.to.call{value: p.amount}("")).
Les appels externes peuvent ouvrir la porte à des attaques de **reentrancy**.

=>Le risque est correctement mitigé grâce à nonReentrant

```
}

    function executeProposal(uint256 _proposalId) external nonReentrant onlyMember {    infinite gas
        Proposal storage p = proposals[_proposalId];
```

b-

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 87:22:

block.timestamp can be influenced by miners

=>Le block.timestamp représente la date et l'heure du bloc miné, mais il peut être légèrement modifié par les mineurs. Ici, il sert seulement à gérer les délais de vote, ce qui est acceptable. Il ne doit pas être utilisé pour des calculs critiques ou sensibles.

c-

Low level calls:

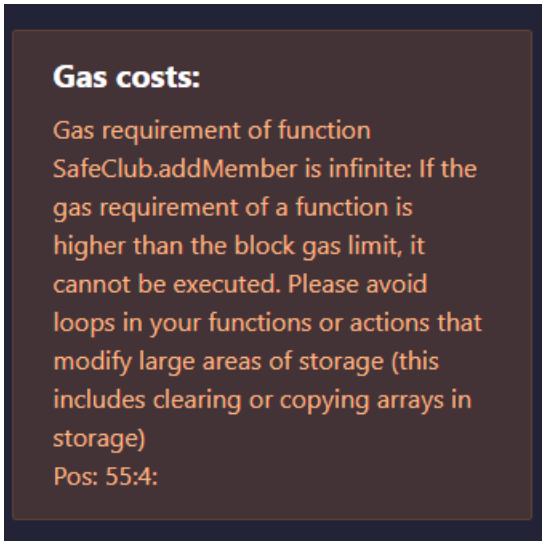
Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
more
Pos: 136:24:

En Solidity, il existe plusieurs manières d'envoyer de l'Ether à une adresse : transfer, send et call. Parmi ces méthodes, call est la plus bas niveau et la plus flexible, mais aussi la plus risquée si elle est mal utilisée.

- **Pourquoi call est risqué ?**
  Parce qu'il exécute un appel externe direct, ce qui peut entraîner des failles de sécurité (comme la réentrance) si on ne contrôle pas bien ce qui se passe après.

=>'utilisation de la fonction low-level call dans le contrat peut présenter des risques de sécurité, notamment des attaques par réentrance. Cependant, dans ce cas, le retour de call est systématiquement vérifié et un modificateur nonReentrant est utilisé pour sécuriser la fonction.

d-



Gas requirement of function is infinite

Analyse :

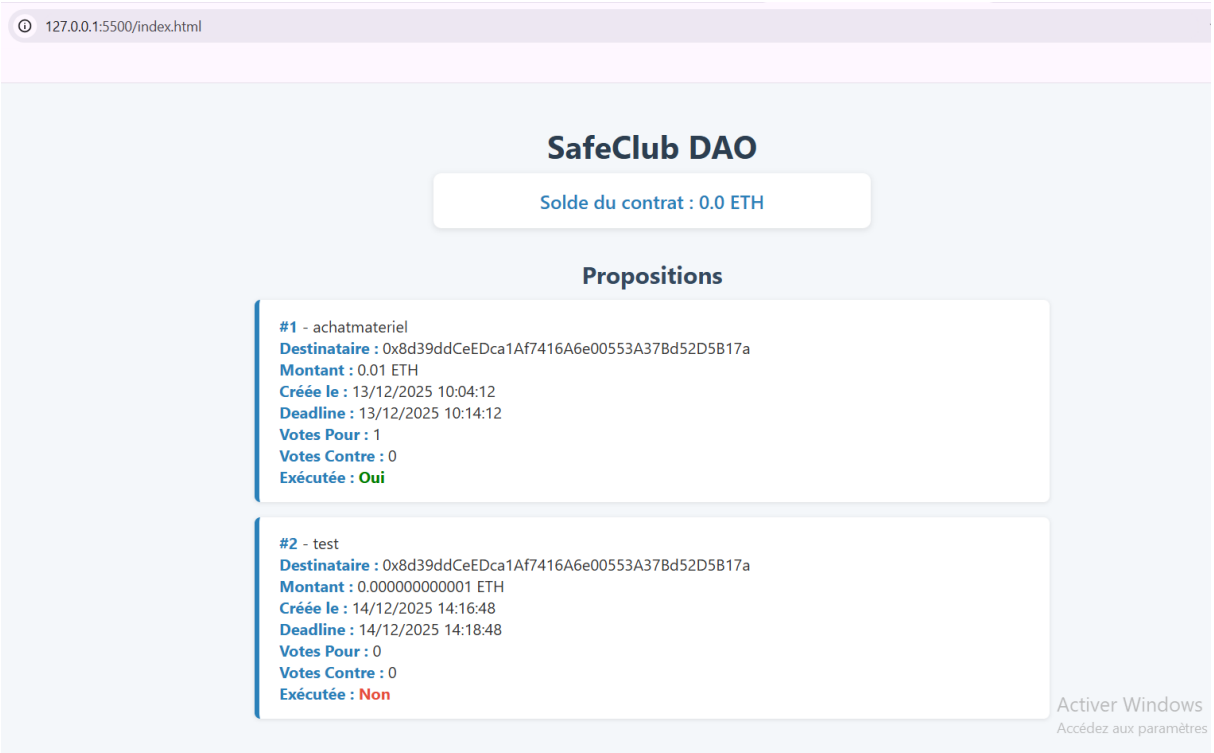
- Aucune boucle
- Aucune itération sur tableaux
- Pas de nettoyage massif du storage

Interface Web

Interface Utilisateur

- Le site affiche clairement le solde total d'Ether détenu par le contrat.
- Il liste toutes les propositions avec les détails suivants :
    - ID de la proposition
    - Description
    - Destinataire de la transaction
    - Montant en Ether
    - Date de création et deadline
    - Nombre de votes pour et contre
    - Statut d'exécution (oui/non)

Technologies utilisées

- **Ethers.js** pour interagir avec le contrat depuis le navigateur.
- **HTML/CSS/JavaScript** pour la construction de l'interface utilisateur.

=>Le site web permet une interaction facile avec la blockchain