SUP'COM

المدرسة العليا للمواصلات بتونس

ÉCOLE SUPÉRIEURE DES COMMUNICATIONS DE TUNIS

Compte rendu du projet de Blockchain

ICO - Plateforme de vente d'objets artisanaux

Réalisé par

Hediene Bahri

Année universitaire : 2022/2023

Idée du projet :

L'idée de mon ICO consiste à concevoir une plateforme qui permet aux utilisateurs d'acheter et de vendre des objets artisanaux fabriqués main. La plateforme pourrait émettre sa propre cryptomonnaie, qui pourrait être utilisée pour acheter et vendre les objets artisanaux. Les utilisateurs qui souhaitent les acheter pourraient donc utiliser la cryptomonnaie pour les acheter directement auprès des artisans, et les artisans pourraient utiliser la cryptomonnaie pour vendre leurs objets. Cette idée vient de l'ambition d'encourager les talentueux de faire émerger leurs produits et assure également une meilleure ouverture sur les différentes cultures des différents pays et permet donc de construire une relation plus solide entre la personne et son identité culturelle.

L'ICO qui va être conçu pourrait être utilisé pour financer le développement de cette plateforme et l'émission de la cryptomonnaie. Les investisseurs de l'ICO pourraient recevoir des tokens qui représentent une part dans la plateforme et une prétention sur une partie des bénéfices.

Comme il est exigé, l'ICO s'étale sur deux périodes (une semaine, et un mois et demi) et pendant la première période, le prix du token est réduit de 22%.

Environnement de développement Ethereum :

- Langage de programmation solidity pour développer des contrats.
- Remix l'IDE en ligne le plus complet pour Solidity.
- Réseau de test Goerli pour le déploiement et les tests des fonctionnalités des contrats
- Metamask.

Le travail établi :

Le travail établi consiste à mettre en place deux contrats *contract HandmadeCradtsToken (ERC20)* et *contract ICO* qui hérite les fonctions de cette token. Tout d'abord, il fallait définir les fonctions relatives au standard ERC20 :

```
4   interface ERC20 {
5       function totalSupply()  external returns (uint256);
6       function balanceOf(address account) external returns (uint256);
7       function transfer(address to, uint256 value) external returns (bool);
8       function transferFrom(address from, address to, uint256 value) external returns (bool);
9       function approve(address spender, uint256 value) external returns (bool);
10      function allowance(address owner, address spender) external view returns (uint256);
11      event Transfer(address indexed from, address indexed to, uint256 value);
12      event Approval(address indexed owner, address indexed spender, uint256 value);
13  }
```

Ensuite, on crée le contrat *HandmadeCradtsToken,* on spécifie le propriétaire du contrat, le nombre total des tokens et on crée un mapping balances qui attribue à chaque adresse une valeur représentant le nombre de tokens qu'elle possède.

```
16  contract HandmadeCraftsToken  {
17
18      address public owner;
19      uint256 public totalSupply;
20      mapping(address => uint256) public balances;
```

Ensuite, on définit le constructeur pour initialiser le contrat et on déclare les attributs suivants : le owner correspond à l'adresse qui a déployé le contrat, le totalSupply est 100000000 et enfin on définit le balances de owner à la valeur de totalsupply :

```
22      constructor() public {
23          owner = msg.sender;
24          totalSupply = 100000000;
25          balances[owner] = totalSupply;
26      }
```

On crée maintenant une fonction *issueToken* qui va permettre à le *owner* de générer plus de tokens et donc augmenter les valeurs de *totalsupply* et *balances* du *owner :*

```
29      function issueTokens(uint256 _amount) public {
30
31      require(msg.sender == owner, "Only the owner can issue tokens.");
32      totalSupply += _amount;
33      balances[owner] += _amount;
34      }
```

On définit la fonction *buyCraft* qui permet aux utilisateurs d'acheter des objets artisanaux en utilisant des tokens. On doit tout d'abord vérifier si l'utilisateur a le nombre nécessaire de tokens pour achever l'achat, puis vérifier si le vendeur possède encore des produits pour les vendre. Une fois ces deux étapes sont établies. un échange de tokens et de produit aura lieu.

```
36      function buyCraft(address _seller, uint256 _price) public payable {
37          require(balances[msg.sender] >= _price, "Insufficient balance.");
38          require(balances[_seller] >= 1, "Seller does not have enough crafts to sell.");
39          balances[msg.sender] -= _price;
40          balances[_seller] += _price;
41          balances[_seller] -= 1;
42          balances[msg.sender] += 1;
43      }
```

D'autre part, on crée la fonction *sellCraft* qui va permettre à des utilisateurs de vendre leurs propres produits qu'ils ont fabriqué. Dans cette étape, il faut vérifier qu'il possède des produits artisanaux à vendre, et puis vérifier que la personne qui va tenter d'acheter le produit spécifié possède le nombre de tokens nécessaire. Puis, un échange de tokens et de produit s'effectue entre les deux parties.

```
45      function sellCraft(address _buyer, uint256 _price) public {
46          require(balances[msg.sender] >= 1, "Seller does not have any crafts to sell.");
47          require(balances[_buyer] >= _price, "Buyer does not have enough tokens.");
48          balances[_buyer] -= _price;
49          balances[msg.sender] += _price;
50          balances[msg.sender] -= 1;
51          balances[_buyer] += 1;
52      }
53  }
```

Après la création du Token, on commence par créer le ICO qui hérite les fonctions du contrat *HandMadeCraftsToken*, on déclare les variables suivants : le prix du token, la date de début de l'ICO, la date de fin de l'ICO et puis l'événement *TokenPurchase :*

```
56  contract ICO is HandmadeCraftsToken {
57
58      uint256 public tokenPrice;
59      uint256 public startTime;
60      uint256 public endTime;
61      event TokenPurchase(address purchaser, uint256 value, uint256 tokens);
```

Le constructeur initialise l'*ICO* en définissant le prix du token (1 wei), la date de début de l'ICO qui est la date de déploiement de celui-ci, et la date de fin qui est fixée à 7 semaines après le début :

```
62      constructor(uint256 _tokenPrice, uint256 _startTime, uint256 _endTime) public HandmadeCraftsToken() {
63      tokenPrice = 1 wei;
64      startTime = block.timestamp;
65      endTime = block.timestamp + 7 weeks;
66  }
```

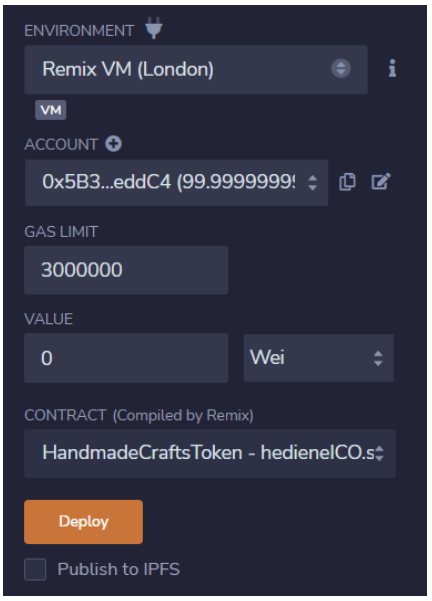
Puis, on crée la derniere fonction de tout l'ICO qui est *buyTokens* qui permet à un utilisateur d'acheter des tokens de la plateforme de vente de produits artisanaux avec de l'argent. La fonction prend un argument *value* qui représente la valeur en éther que l'utilisateur souhaite investir dans les tokens. Cette fonction vérifie d'abord si l'ICO est en cours en comparant l'horodatage actuel avec les horodatages de début et de fin de l'ICO. Si l'ICO est en cours, la fonction calcule le nombre de tokens que l'utilisateur recevra en fonction de la valeur qu'il investit et du prix des tokens.

Si l'horodatage actuel est dans la première semaine de l'ICO, la fonction applique une remise de 22% sur le prix de tokens. La fonction vérifie ensuite si l'utilisateur a suffisamment d'éther pour acheter les tokens et met à jour les balances de l'utilisateur et de la totalité des tokens. Enfin, la fonction émet l'événement *TokenPurchase* avec les informations sur l'achat de tokens.

```
68      function buyTokens(uint256 value) public payable {
69
70          require(block.timestamp >= startTime && block.timestamp <= endTime, "ICO is not ongoing.");
71          uint256 tokens = (value*(10 ** 18)) / (tokenPrice);
72          if (block.timestamp <= startTime + 1 weeks) {
73              uint256 discount = (tokens*(22)) / (100);
74              tokens = tokens - discount;
75          }
76
77          require(value >= (tokens*(tokenPrice)) / (10 ** 18), "Insufficient balance.");
78          balances[msg.sender] = balances[msg.sender] + tokens;
79          totalSupply = totalSupply - tokens;
80          msg.sender.transfer(tokens);
81          emit TokenPurchase(msg.sender, value, tokens);
82      }
83  }
```

- **Déploiement du contrat sur Remix :**

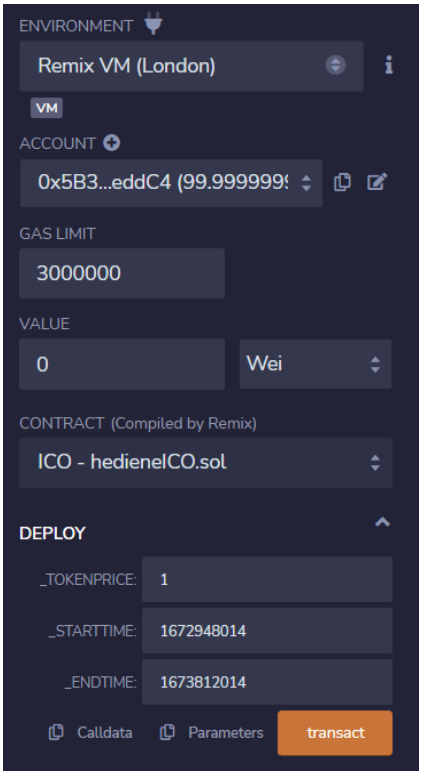
On déploie tout d'abord le contrat *HandMadeCraftsToken* sur le Remix VM pour le test du code :



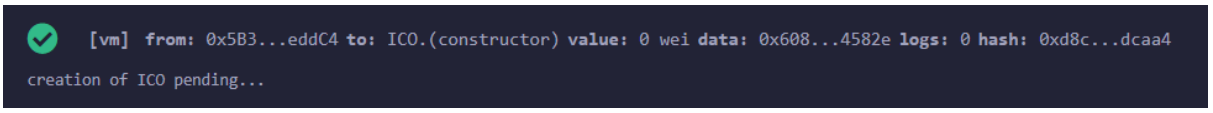
Le contrat a été bien deployé :

```
[vm] from: 0x5B3...eddC4 to: HandmadeCraftsToken.(constructor) value: 0 wei data: 0x608...60033 logs: 0 hash: 0xde5...9c8bb
```

Ensuite, on déplace le Contrat ICO sur le même environnement, on spécifie le prix du Token, et horodatages de début et de fin selon l'horodatage du système Unix Epoch pour faciliter le processus :

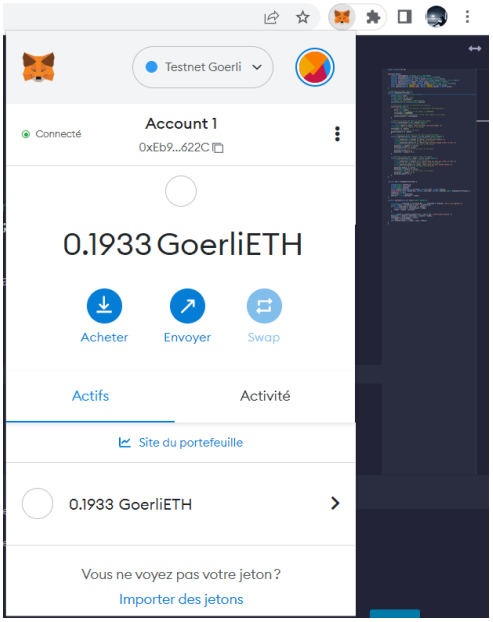


L'ICO a été bien déployé :

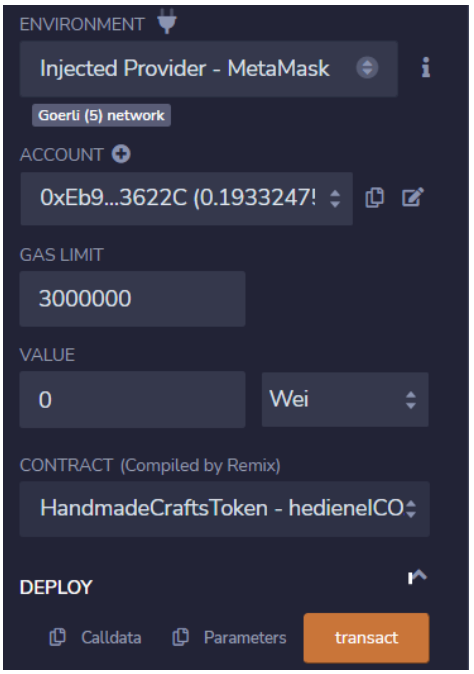


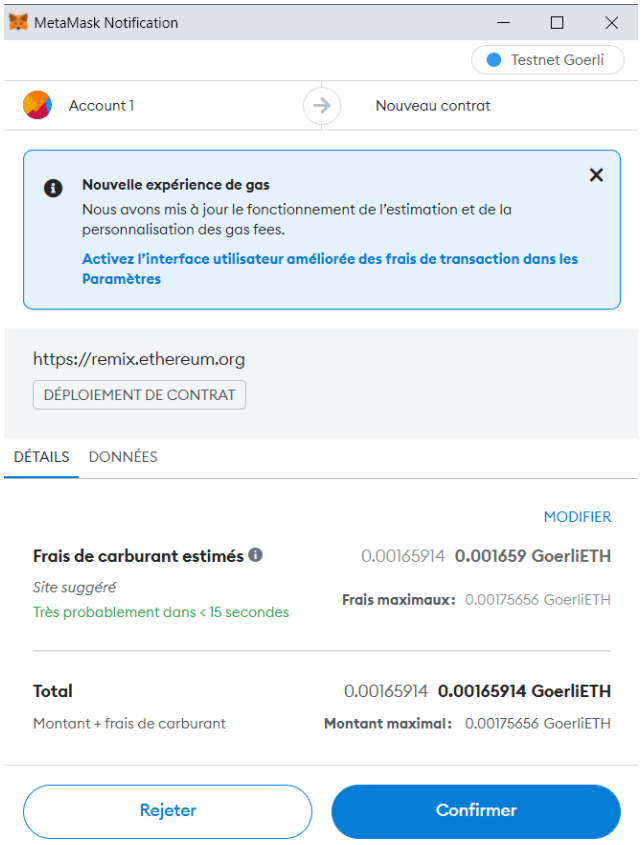
- **Déploiement du contrat avec Netmask :**

On vérifie l'état de Metamask, il est "connecté", le montant d'éther est suffisant pour le test :

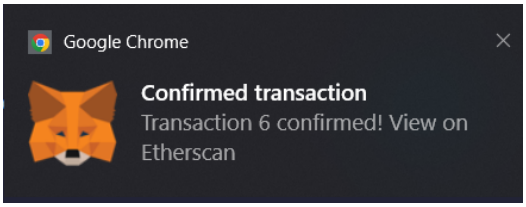


On déploie contrat *HandMadeCraftsToken* et on confirme les transactions sur metamask :

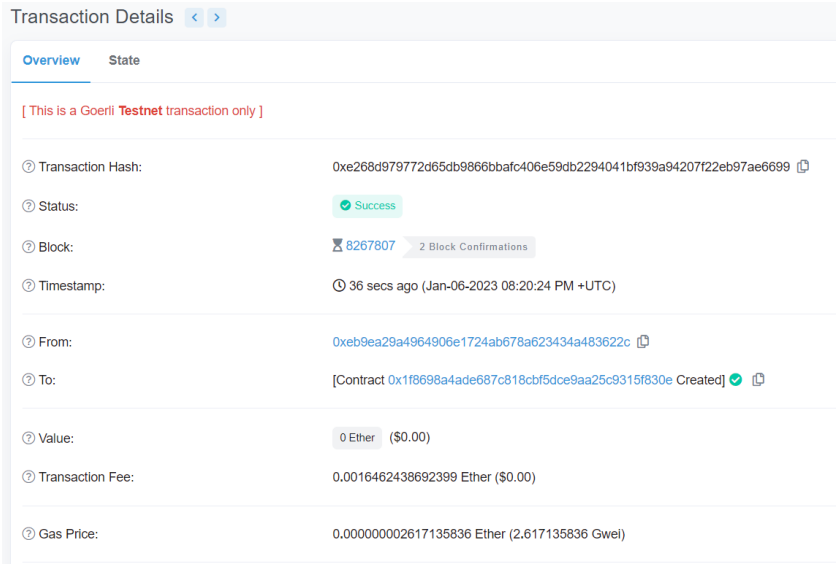




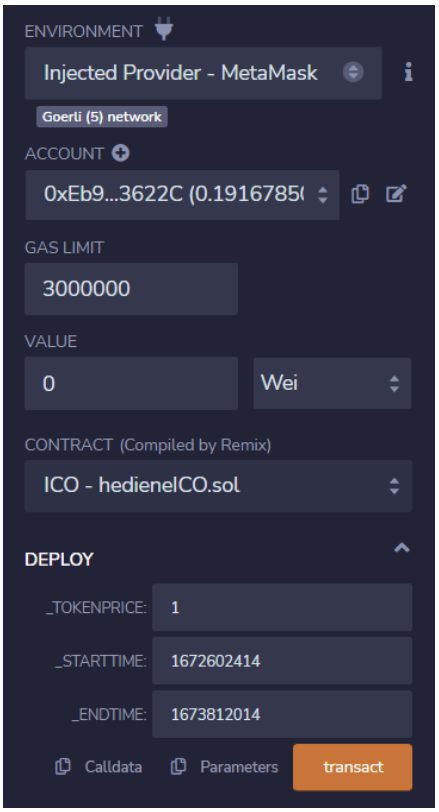
Après la réussite de déploiement, une notification s'affiche pour confirmer la transaction effectuée :



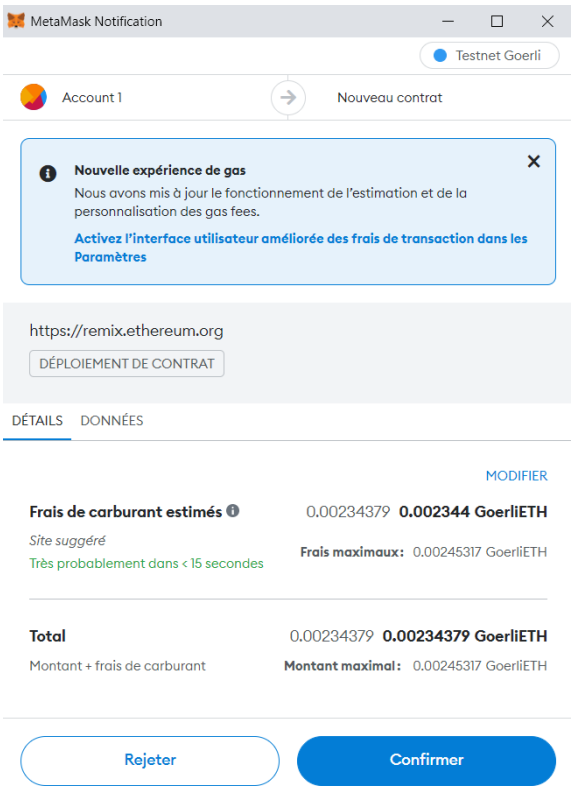
On peut même voir plus de détails sur la transaction au niveau du site etherscan :



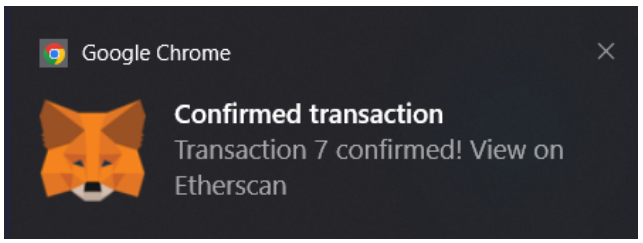
Maintenant, on déploie le *ICO* et on met en entrée les mêmes valeurs mises au niveau du test sur l'environnement Remix :



On confirme la transaction sur Metamask :



et la notification de confirmation s'affiche :



Sur le site Etherscan, on peut visualiser tous les contrats et transactions déployés :