



UNIVERSITE ABDELMALEK ESSAADI

Master IA et science de données

Module : BlockChain

Atelier 5 : Mini Social Média

Réalisé PAR :

Aymane Mahri

Encadré PAR :

Pr. Ikram Ben abdel ouahab

Table des matières :

Introduction :	3
1 - Déclaration du Smart Contract :	4
2 – Liste des messages :	5
3 – Fonction de Publication :	6
4 – Fonction de Consultation :	6
5 – Fonction pour récupérer le nombre de messages :	7
6 – Déploiement et Tests :	7
Conclusion :	11

Introduction :

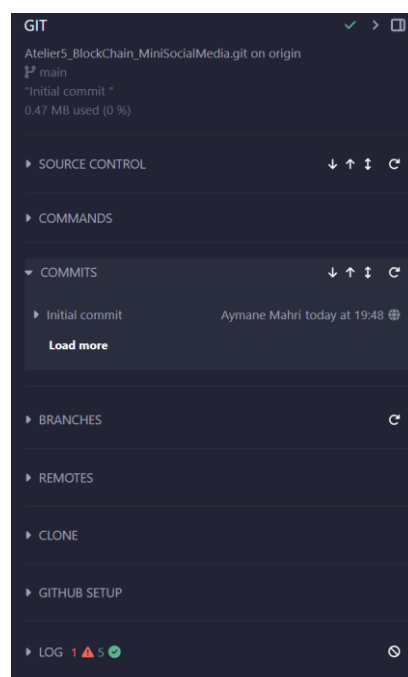
Dans le cadre de cet atelier, nous avons développé un mini réseau social décentralisé sous forme de smart contract en utilisant le langage Solidity sur la blockchain Ethereum. L'objectif de ce projet est d'implémenter une plateforme de publication de messages où les utilisateurs peuvent poster des messages publics et consulter les publications existantes. Ce type d'application décentralisée (DApp) présente des avantages uniques comme l'immutabilité, la transparence et l'absence d'intermédiaires, des aspects essentiels des technologies blockchain. Pour cette application, nous avons choisi d'utiliser **Remix IDE** pour la programmation et le déploiement, associé à **MetaMask** pour l'interaction avec le réseau de test **SepoliaETH**.

Le rapport décrit en détail la structure et le fonctionnement du smart contract, les étapes de son développement, les tests effectués et les résultats obtenus, afin de valider son bon fonctionnement.

1 - Déclaration du Smart Contract :

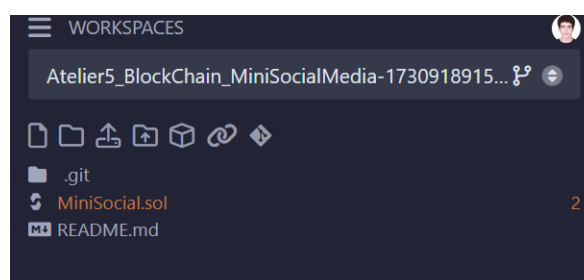
Nous commençons tout d'abord par créer un nouveau répertoire GitHub ensuite nous nous connectons à GitHub depuis remix IDE en l'autorisant à accéder à notre compte GitHub, après il suffit juste de cloner le répertoire de GitHub que nous avons créé :

Si tout cela se passe bien voici l'interface que nous devrions avoir :



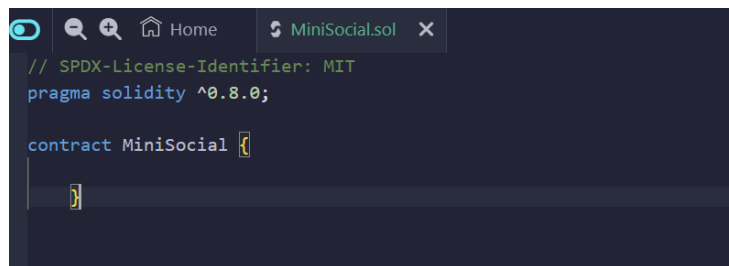
Maintenant après que cela soit fait nous pouvons passer à notre partie code pour commencer notre mini social média.

On Commence tout d'abord par créer un fichier appelée **MiniSocial.Sol** :



L'erreur est simplement dû au fait que le fichier est toujours vide il faut qu'on spécifie la License et le compiler solidity.

Maintenant nous allons spécifier la license et le compiler solidity et définir le smart contract **MiniSocial** :



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract MiniSocial {
```

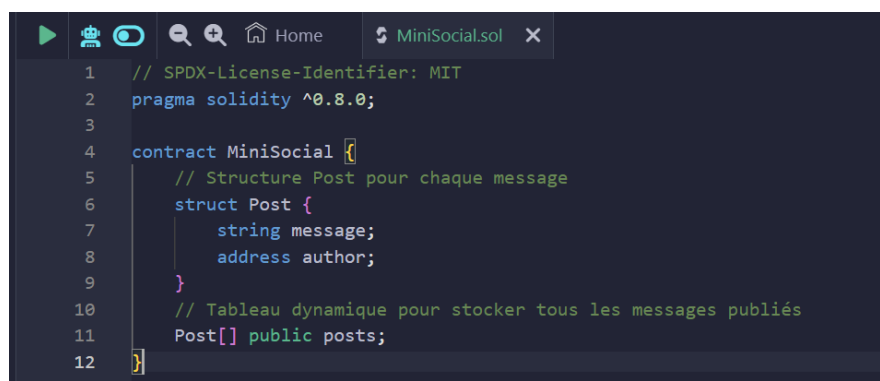
Maintenant nous allons déclarer une structure **Post** qui contiendra un message de type string et l'adresse author de l'utilisateur qui a publié le message, et cela à l'intérieur du smart contract **MiniSocial** :



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract MiniSocial {
5     // Structure Post pour chaque message
6     struct Post {
7         string message;
8         address author;
9     }
10 }
```

2 – Liste des messages :

Pour faire cela, nous allons déclarer un tableau dynamique *posts* de type **Post[]** pour stocker tous les messages publiés :



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract MiniSocial {
5     // Structure Post pour chaque message
6     struct Post {
7         string message;
8         address author;
9     }
10     // Tableau dynamique pour stocker tous les messages publiés
11     Post[] public posts;
12 }
```

3 – Fonction de Publication :

Maintenant, nous allons ajouter une fonction **publishPost** qui permet aux utilisateurs de publier un message. Cette fonction crée un nouveau **Post** avec le message et l'adresse de l'utilisateur (`msg.sender`) et l'ajoute au tableau **posts** :

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract MiniSocial {
5     // Structure Post pour chaque message
6     struct Post {
7         string message;
8         address author;
9     }
10    // Tableau dynamique pour stocker tous les messages publiés
11    Post[] public posts;
12
13
14    // Fonction pour publier un nouveau message
15    function publishPost(string memory _message) public { infinite gas
16        // Crée un nouveau post et l'ajoute au tableau des posts
17        posts.push(Post(_message, msg.sender));
18    }
19
20
21 }
```

4 – Fonction de Consultation :

Ensuite, nous allons ajouter une fonction **getPost** qui prend un index en paramètre pour récupérer le message et l'auteur du post correspondant :

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract MiniSocial {
5     // Structure Post pour chaque message
6     struct Post {
7         string message;
8         address author;
9     }
10    // Tableau dynamique pour stocker tous les messages publiés
11    Post[] public posts;
12
13
14    // Fonction pour publier un nouveau message
15    function publishPost(string memory _message) public { infinite gas
16        // Crée un nouveau post et l'ajoute au tableau des posts
17        posts.push(Post(_message, msg.sender));
18    }
19
20    // Fonction pour consulter un post par index
21    function getPost(uint index) public view returns (string memory, address) { infinite gas
22        require(index < posts.length, "Index hors limite"); // Vérifie si l'index est valide
23        Post storage post = posts[index];
24        return (post.message, post.author);
25    }
26 }
```

5 – Fonction pour récupérer le nombre de messages :

Pour faire cela, nous allons ajouter une fonction **getTotalPosts** pour retourner le nombre total de messages publiés :

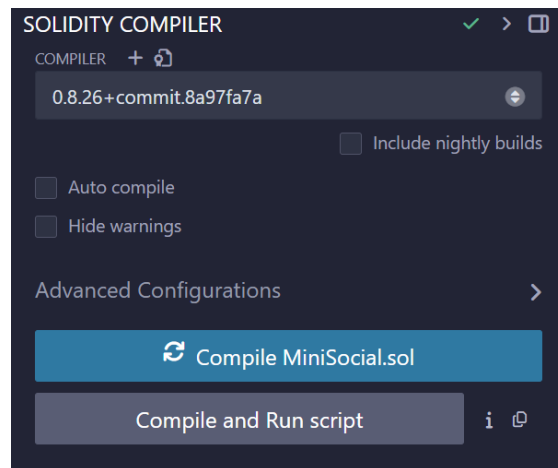
```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract MiniSocial {
5     // Structure Post pour chaque message
6     struct Post {
7         string message;
8         address author;
9     }
10    // Tableau dynamique pour stocker tous les messages publiés
11    Post[] public posts;
12
13
14    // Fonction pour publier un nouveau message
15    function publishPost(string memory _message) public {
16        // Crée un nouveau post et l'ajoute au tableau des posts
17        posts.push(Post(_message, msg.sender));
18    }
19
20    // Fonction pour consulter un post par index
21    function getPost(uint index) public view returns (string memory, address) {
22        require(index < posts.length, "Index hors limite"); // Vérifie si l'index est valide
23        Post storage post = posts[index];
24        return (post.message, post.author);
25    }
26
27    // Fonction pour obtenir le nombre total de messages publiés
28    function getTotalPosts() public view returns (uint) {
29        return posts.length;
30    }
31 }
```

6 – Déploiement et Tests :

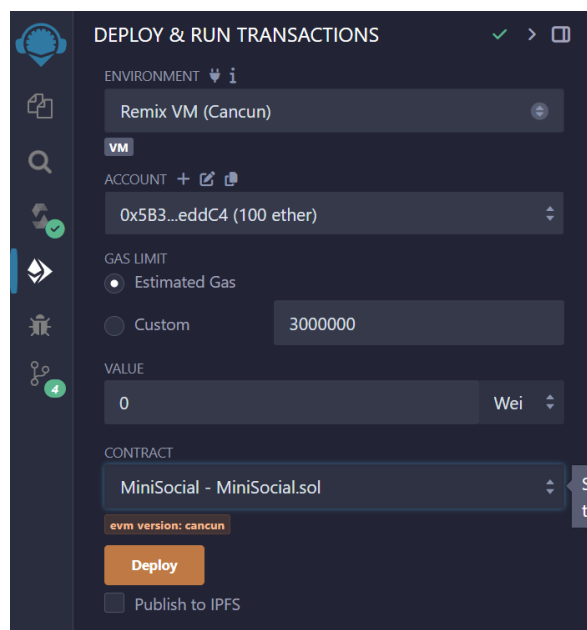
Mainetenat, après avoir fait tout cela et terminé notre code, nous pouvons passer au déploiement et test de notre mini réseau social.

(On choisi comme environnement par défaut VM pour tester car pour utiliser Metamask on devra avoir déjà des Eth dans notre wallet)

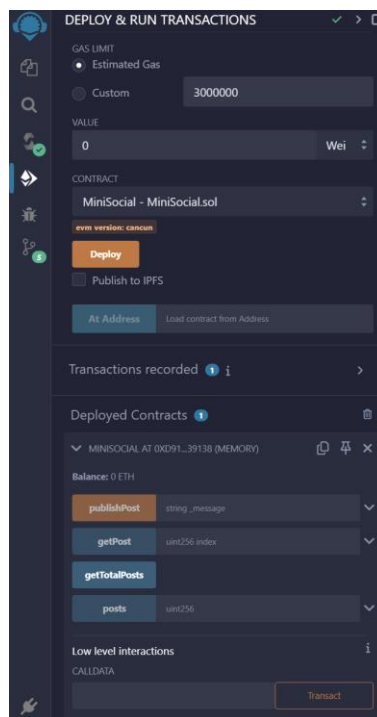
Tout d'abord on commence par compiler notre fichier **MiniSocial.sol** :



Ensuite après avoir compiler notre fichier, il faudra maintenant passer à son déploiement et c'est exactement ce que nous allons faire dans la partie **Deploy & run transactions** de remix IDE :

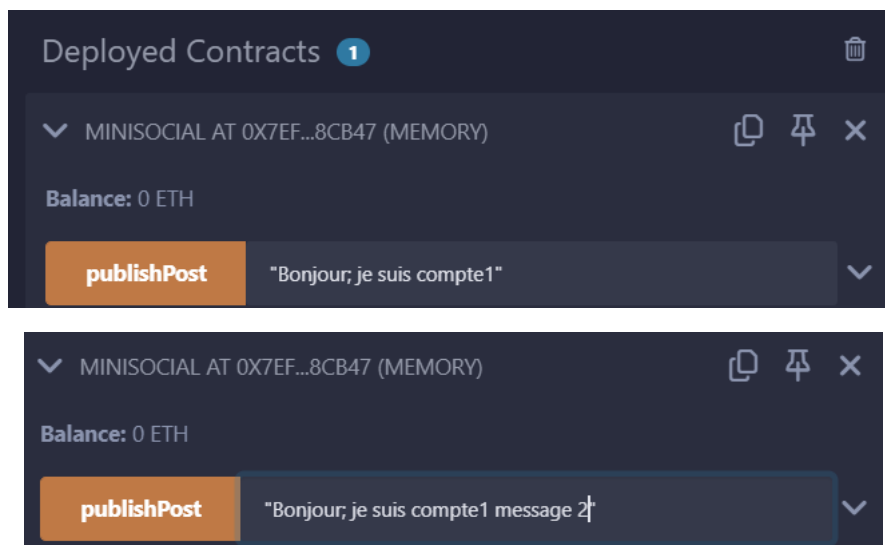


Après avoir fait cela, nous pourrons enfin checker notre contract déployé dans la rubrique Deployed Contracts et le tester :



Maintenant commençons notre test, nous allons tout d'abord dans notre premier compte publier un post et cela en écrivant ce qu'on veut et ensuite en cliquant sur **PublishPost** :

Nous allons publier 2 messages :



Notre publication se fait avec succès :

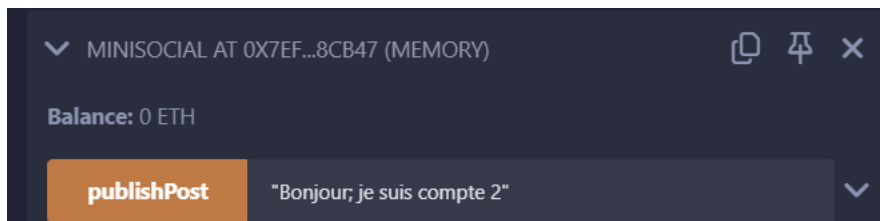
```
transact to MiniSocial.publishPost pending ...  
  
[vm] from: 0x5B3...eddC4 to: MiniSocial.publishPost(string) 0x7EF...8CB47 value: 0 wei data: 0xc57...00000 logs: 0 hash: 0x6a2...d3f65
```

Essayons Maintenant de checker le nombre des posts publiés en cliquant sur **getTotalPosts** :

```
getTotalPosts  
  
0: uint256: 2
```

Elle nous retourne 2 posts publiés avec succès.

Ajoutons maintenant un autre post avec un autre compte :

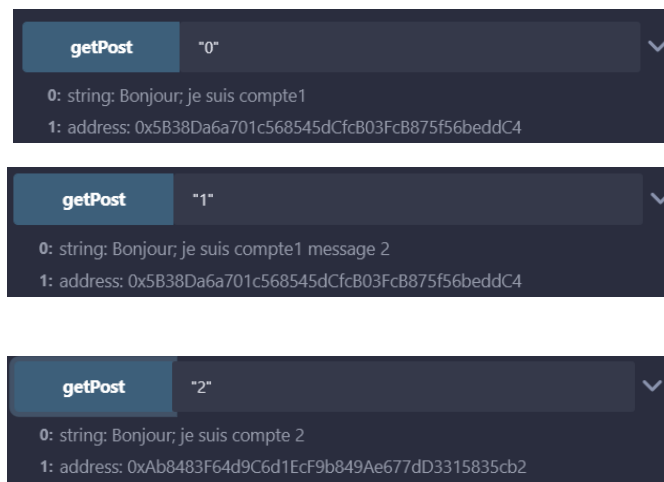


Maintenant, recheckons le nombre de posts publiés en cliquant sur **getTotalPosts** :

```
getTotalPosts  
  
0: uint256: 3
```

Effectivement, le post a été ajouté donc on a 3 posts en total.

Maintenant essayons de consulter post par post en indiquant l'index du post :



On peut remarquer que chaque index rend son post respectif, on a les 2 premiers posts publiés avec les index 0 et 1, qui ont été publiés par le même compte, et c'est exactement ce qu'on peut constater, même chose avec le 3eme post publié qui a été publié par un autre compte.

Donc on peut conclure que notre mini réseau social marche avec succès.

Conclusion :

Ce projet a permis d'illustrer les concepts fondamentaux de la programmation de smart contracts avec Solidity et de démontrer comment construire une application décentralisée basique de type réseau social. Le smart contract **MiniSocial** remplit les fonctionnalités essentielles d'une plateforme sociale en permettant aux utilisateurs de publier et de consulter des messages en toute transparence et sans intermédiaires. Grâce aux tests, nous avons validé le bon fonctionnement des fonctionnalités et assuré que le contrat répond aux exigences.

Ce projet ouvre la voie à des extensions potentielles, telles que l'ajout de réactions, de commentaires ou de fonctionnalités de suivi, qui pourraient enrichir l'expérience utilisateur et illustrer encore davantage le potentiel des applications décentralisées dans le domaine des réseaux sociaux.