1/2 Module : Blockchain
2ème année Master IASD
2024/2025

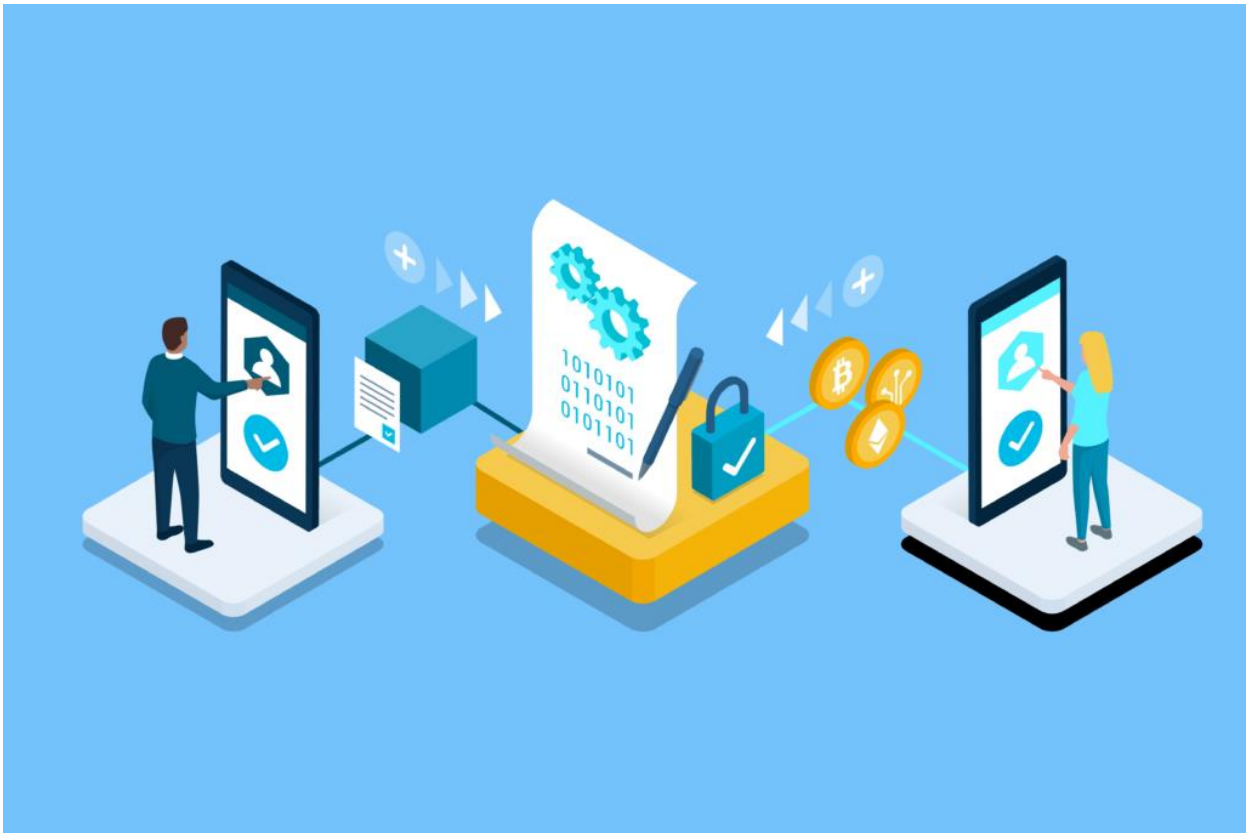
Réalisé par : Ammar AMZIL

Encadré par : Pr. Ikram BENADBELOUAHAB

ATELIER 5 : SMART CONTRACT "MINI RESEAU SOCIAL" EN SOLIDITY

1. Introduction :

Les contrats intelligents, ou smart contracts, sont des programmes informatiques qui résident sur une blockchain et s'exécutent automatiquement dès que des conditions préalablement définies sont remplies. Grâce à des langages comme Solidity, ces contrats permettent d'automatiser des transactions et des interactions sans besoin d'intermédiaires, offrant ainsi une grande transparence et sécurité. Bien qu'ils soient principalement utilisés dans des environnements financiers, leur application s'étend à divers secteurs, y compris les réseaux sociaux. Dans ce cadre, les smart contracts peuvent décentraliser la gestion des données des utilisateurs et des interactions. Par exemple, ils peuvent gérer la publication de contenu, assurer la confidentialité des informations ou encore récompenser automatiquement les utilisateurs pour leurs actions. Cette approche offre des solutions aux défis rencontrés par les réseaux sociaux traditionnels, en particulier en matière de contrôle des données et de monétisation des interactions.



Lors de cet atelier, nous simulerons des opérations de gestion de publications sur les réseaux sociaux à l'aide de smart contracts. Nous utiliserons Remix IDE, un environnement de développement pour Solidity, ainsi que MetaMask, un portefeuille de crypto-monnaies, pour interagir avec la blockchain. Cette simulation permettra d'explorer le potentiel des smart contracts dans les réseaux sociaux et d'apprendre les bases de la programmation sur blockchain.

2. Solidity :

Solidity est un langage de programmation orienté objet, similaire à des langages comme JavaScript, Python ou C++. Il est spécialement conçu pour créer des smart contracts sur la blockchain Ethereum, ainsi que sur d'autres blockchains compatibles, telles que Tomochain. Afin de faciliter son adoption par les développeurs et d'aider les débutants à l'apprendre facilement,

Solidity s'inspire de plusieurs langages populaires comme JavaScript, C++ et C#. Le projet a été proposé par Gavin Wood, cofondateur et ancien CTO d'Ethereum, et son développement a commencé en 2014 au sein de l'équipe d'Ethereum. Solidity fait partie des quatre langages permettant d'interagir avec l'EVM (Ethereum Virtual Machine). Il est largement utilisé pour développer des dApps et a également joué un rôle dans des projets comme un Proof of Concept (PoC) de SWIFT sur une blockchain privée, Borrow. Sur le plan technique, Solidity suit le paradigme de la programmation orientée objet, où des objets, qui sont des instances de concepts ou d'entités, interagissent entre eux. Comme beaucoup d'autres langages, il prend en charge la manipulation de fonctions, de structures de données et de chaînes de caractères (strings).

3. Remix IDE :

Remix IDE est un environnement de développement intégré (IDE) conçu pour la création, le débogage et le déploiement de smart contracts en Solidity sur la blockchain Ethereum. Accessible directement via un navigateur web, il offre une interface intuitive qui permet aux développeurs de coder et de tester leurs contrats intelligents sans nécessiter de configuration complexe. Parmi ses fonctionnalités principales, Remix propose un compilateur personnalisable, un débogueur intégré, un outil pour analyser les transactions et les frais de gas, ainsi que diverses options de déploiement sur la blockchain. Il simplifie également l'utilisation des smart contracts OpenZeppelin et inclut un mode sombre pour plus de confort. Toutes ces fonctionnalités sont réunies sur une seule interface web, facilitant ainsi l'expérience de développement.

4. Metamask :

MetaMask est une extension de navigateur et un portefeuille numérique qui permet aux utilisateurs d'accéder facilement aux applications décentralisées (dApps) et de gérer leurs actifs numériques sur la blockchain Ethereum et d'autres réseaux compatibles. Agissant comme une passerelle entre un navigateur traditionnel et le monde de la blockchain, MetaMask permet d'interagir avec des smart contracts, de gérer des cryptomonnaies comme l'Ether (ETH) et les tokens ERC-20, ainsi que d'explorer des dApps directement depuis le navigateur ou l'application mobile. Il offre des outils sécurisés pour stocker, envoyer et recevoir des cryptomonnaies, et

permet un accès fluide à diverses dApps, notamment dans les domaines de la finance décentralisée (DeFi), des jeux blockchain et des marchés de NFT. MetaMask prend en charge plusieurs réseaux, dont Binance Smart Chain et Polygon, et permet une gestion simplifiée des connexions à ces différents réseaux. Côté sécurité, les clés privées sont chiffrées et stockées localement, et chaque transaction ou interaction avec une dApp nécessite une confirmation de l'utilisateur. Compatible avec les navigateurs Chrome, Firefox, Edge, Brave, ainsi qu'avec les plateformes mobiles iOS et Android, MetaMask se distingue par sa facilité d'utilisation et son intégration poussée, ce qui en fait un outil incontournable pour les utilisateurs de la DeFi et des NFT.

5. Travaille à faire :

Ce travail consiste à créer, déployer et tester un smart contract simple appelé "MiniSocial" en utilisant Remix IDE et MetaMask sur le réseau de test SepoliaETH. Le smart contract inclut une structure `Post` pour enregistrer des messages de type string et l'adresse de l'auteur. Un tableau dynamique `posts` est utilisé pour stocker tous les messages publiés.

Le contrat comprend trois fonctions principales :

1. **publishPost**, qui permet aux utilisateurs de publier un message en l'ajoutant au tableau 'posts' avec l'adresse de l'auteur.

```
//Question3: Création de la fonction de publication.
function publishPost(string memory _message) public{    infinite gas
    Post memory new_post = Post({
        message: _message,
        author: msg.sender
});
posts.push(new_post);
}
```

2. **getPost**, qui permet de consulter un message spécifique en fournissant son index, renvoyant le texte du message et l'adresse de l'auteur.

```
//Question4: Création de function de consultation.
function getPost(uint index) public view returns (string memory, address){    infinite gas
    Post storage post =posts[index];
    return (post.message,post.author);
}
```

3. **getTotalPosts**, qui retourne le nombre total de messages publiés.

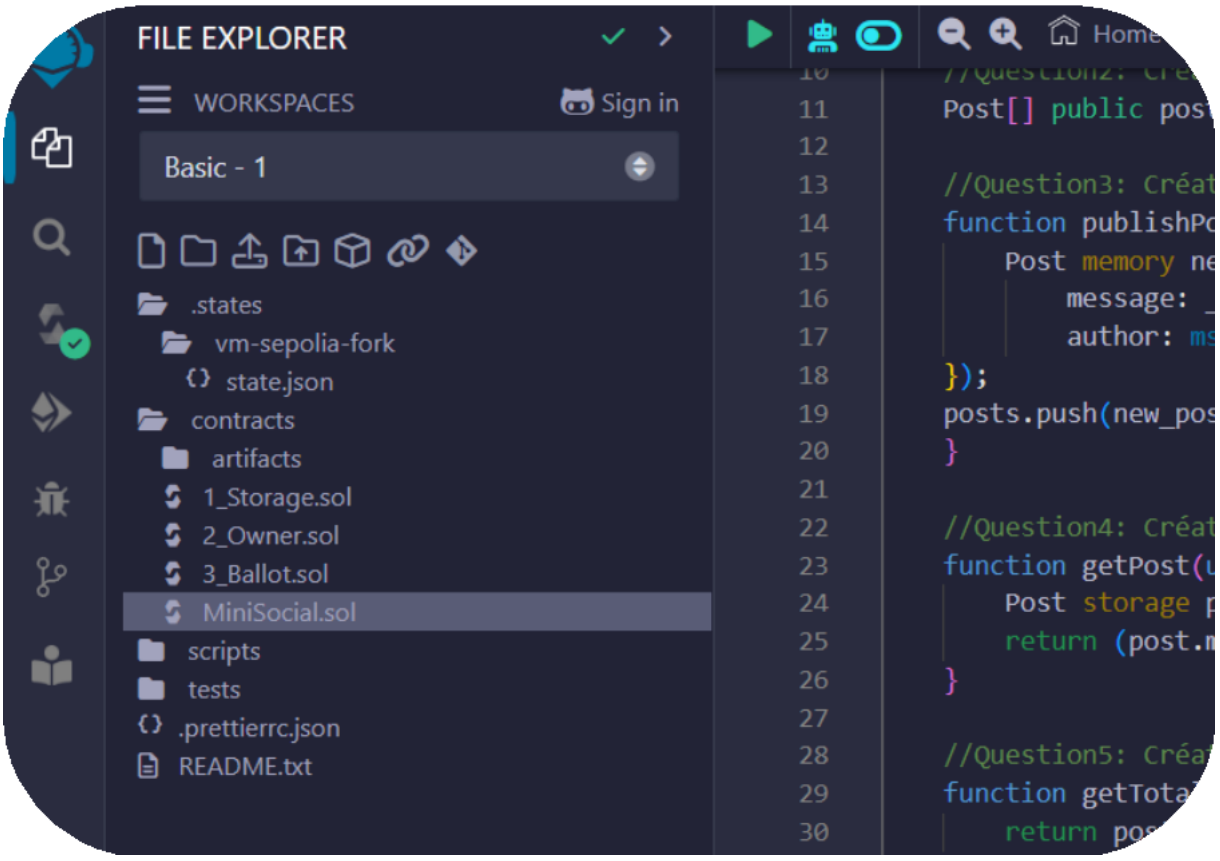
```
28    //Question5: Création de fonction pour récupérer le nombre des messages publiés.
29    function getTotalPosts() public view returns (uint){    2462 gas
30        return posts.length;
31    }
```

Une fois le smart contract rédigé, il est déployé sur un réseau de test via MetaMask. Des tests sont réalisés pour vérifier le bon fonctionnement des fonctions, notamment en publiant des messages avec différentes adresses, en consultant des messages spécifiques et en s'assurant que le nombre total de messages est correctement comptabilisé.

6. Application :

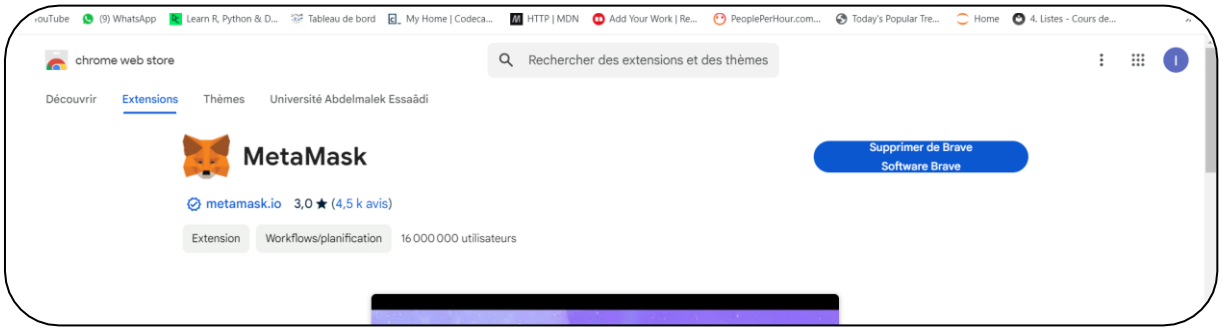
Étape 1 : Déclaration du Smart Contract :

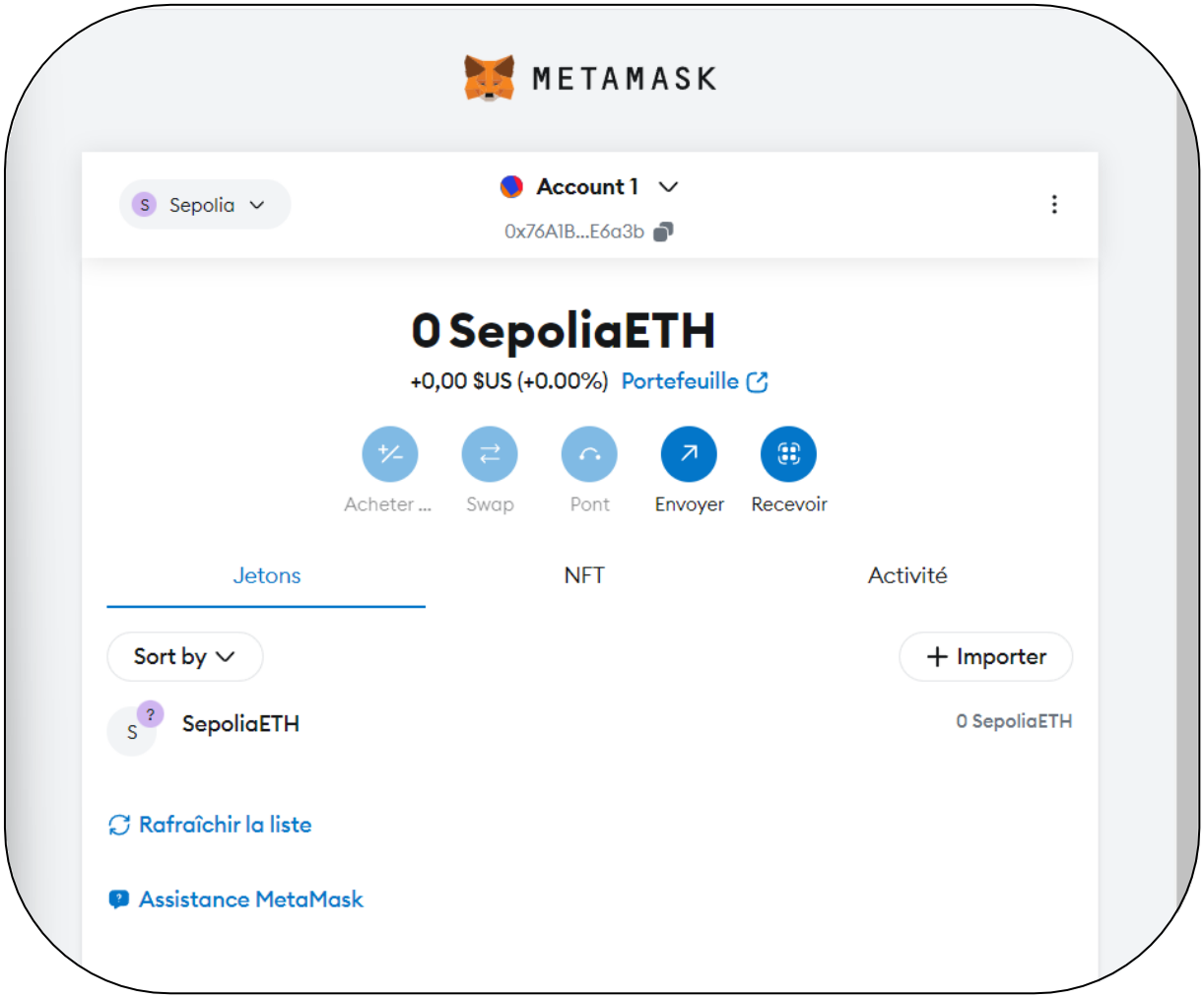
- Ouvrir REMIX IDE dans un navigateur (ce projet compte sur le navigateur BRAVE,https://remix.ethereum.org).
- Créer un fichier MiniSocial.sol
- Écrire le code de la smart contrat avec Solidity dans un fichier *MiniSocial.sol*



Étape 2 : Déploiement et Tests :

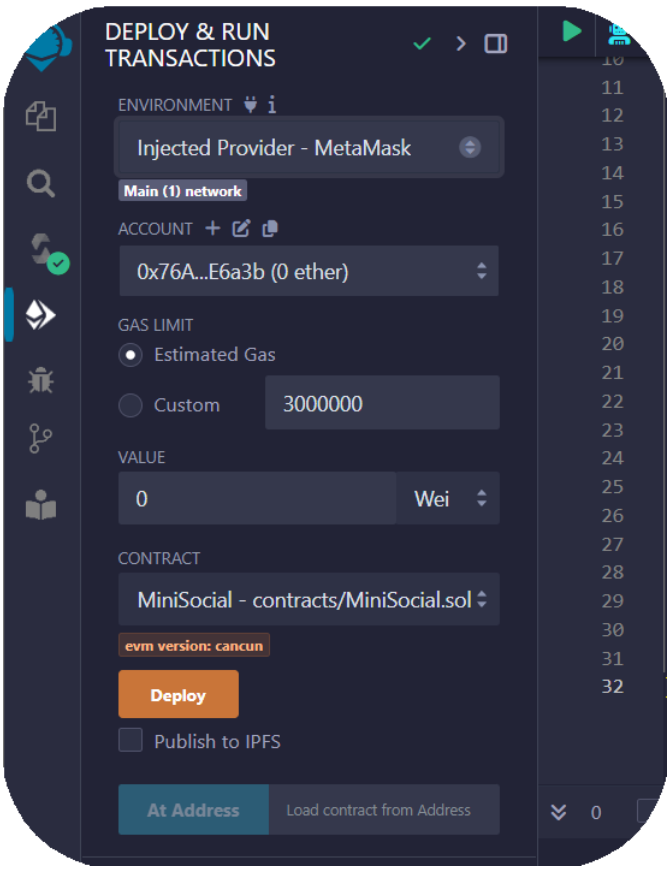
- **Installer l'extension MetaMask** : Assurez que l'extension MetaMask est installée et configurée sur le réseau de test Sepolia.



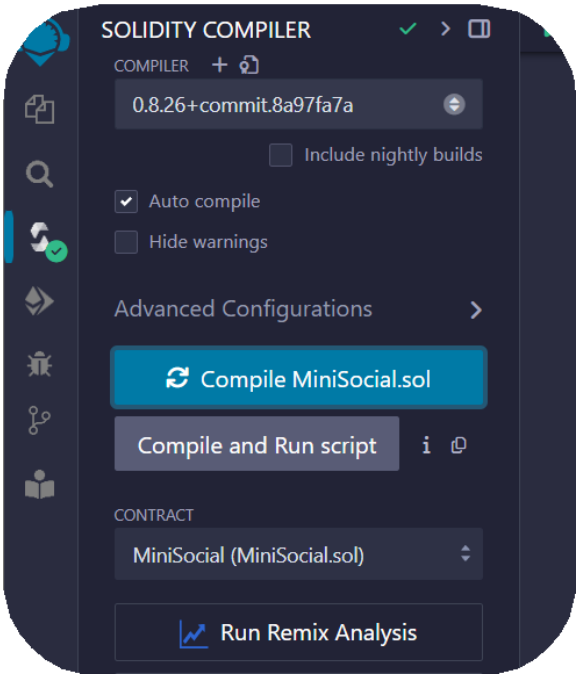


- **Déploiement du contrat**

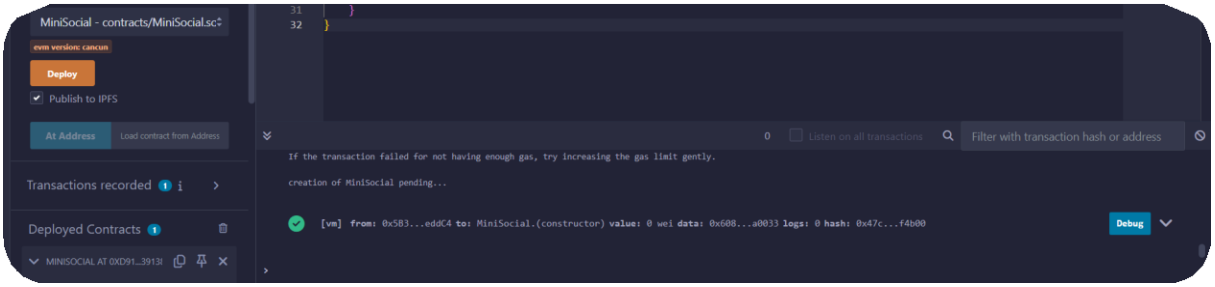
Dans Remix, connectez MetaMask en choisissant le réseau Sepolia dans les paramètres deRemix.

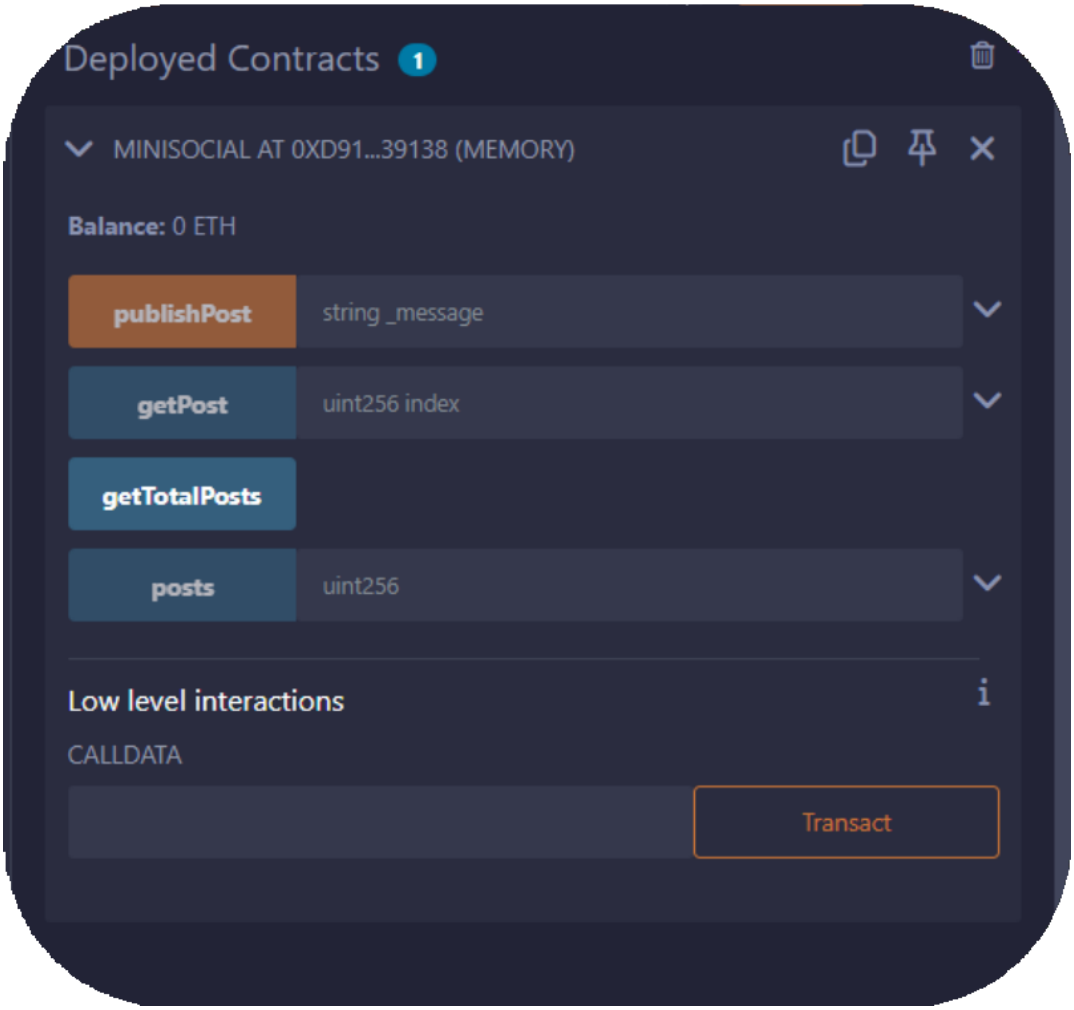


Compilez le contrat (MiniSocial.sol) en sélectionnant la bonne version de Solidity.



Dans l'onglet "**Deploy & Run Transactions**", sélectionnez votre contrat MiniSocial et cliquez sur "Deploy".





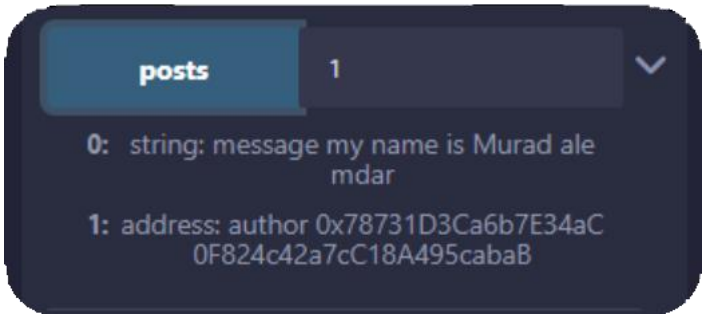
- **Test de la fonction publishPost** :

  Utilisez plusieurs comptes pour appeler publishPost avec différents messages. Chaqueappel enregistre un message et l'adresse de l'utilisateur dans le tableau posts.

- **Test de la fonction getPost** : Appelez getPost en fournissant différents indices pour vérifier que les messages et les adresses des auteurs sont retournés correctement.

- **Test de la fonction getTotalPosts** : Appelez cette fonction pour vérifier que le nombre de messages est correct.

- **Test de le contenu des posts** : Appelez cette fonctionnalité pour vérifier le contenu des messages est correct.



7. Conclusion :

Ce projet a consisté en la création d'un mini réseau social décentralisé, offrant une opportunité d'explorer les fonctionnalités clés des smart contracts en Solidity. Nous avons développé le contrat intelligent MiniSocial, intégrant des fonctions essentielles comme la publication et la visualisation de messages ainsi que le suivi du nombre total de publications. Pour faciliter ce processus, nous avons utilisé Remix IDE pour le déploiement et les tests, et MetaMask pour simuler des transactions sur un réseau de test. Ce projet illustre comment les smart contracts peuvent contribuer à la décentralisation des réseaux sociaux en apportant transparence et immuabilité aux interactions des utilisateurs. Les smart contracts garantissent que chaque message est associé de manière sécurisée à son auteur et accessible publiquement sans dépendre d'une autorité centrale. Cette approche ouvre de nouvelles perspectives pour les applications décentralisées, où les utilisateurs ont un contrôle direct sur leurs données et leurs interactions dans un environnement sécurisé. En somme, ce projet a montré la puissance des smart contracts pour concevoir des systèmes décentralisés et transparents, constituant une base solide pour des applications autonomes et résistantes à la censure, et jetant les bases d'une nouvelle génération de réseaux sociaux et services numériques.