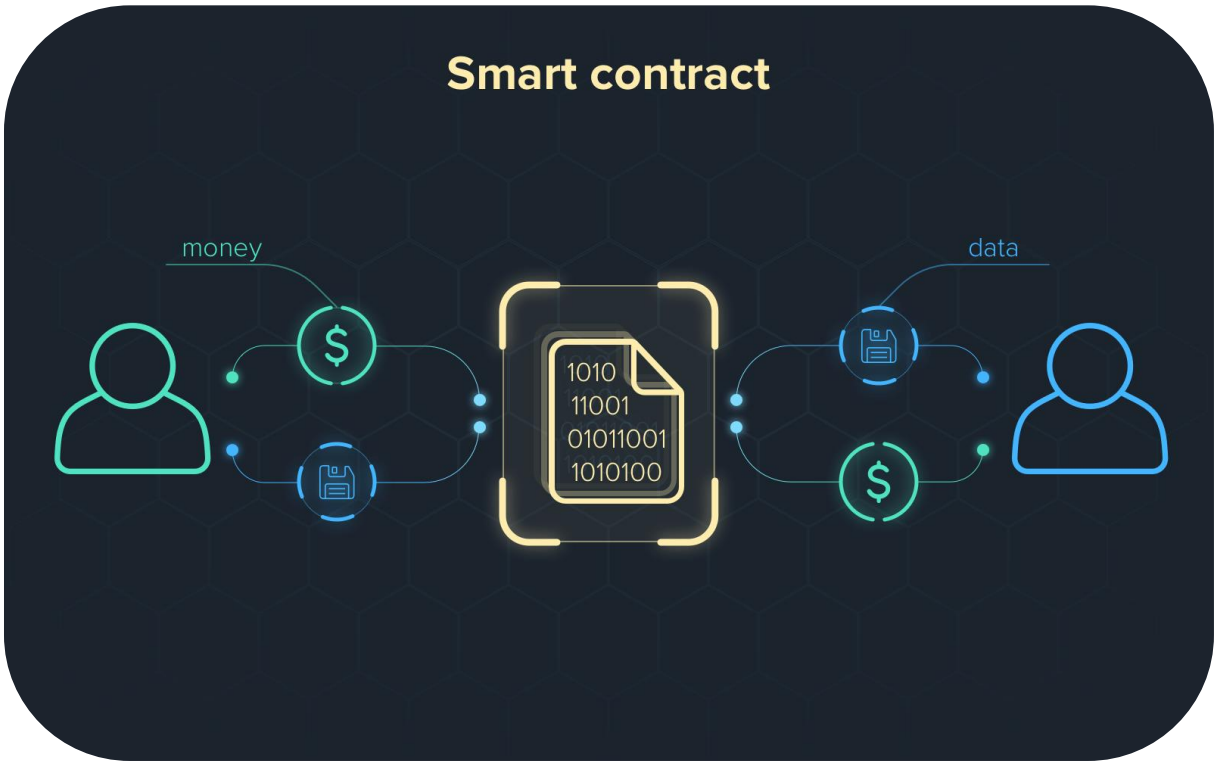
SMART CONTRACT "MINI RESEAU SOCIAL" EN SOLIDITY

1. **Introduction :**

Les *smart contracts*, ou contrats intelligents, sont des programmes informatiques stockés sur une blockchain qui s'exécutent automatiquement lorsque certaines conditions prédéfinies sont remplies. En utilisant des langages comme Solidity, les smart contracts permettent l'automatisation de transactions et d'interactions sans nécessiter d'intermédiaire, offrant ainsi un haut niveau de transparence et de sécurité. Ces contrats numériques sont souvent utilisés dans des environnements financiers, mais ils trouvent également leur place dans de nombreux autres domaines, y compris les réseaux sociaux.



Dans le contexte des réseaux sociaux, les smart contracts permettent de décentraliser la gestion des données des utilisateurs et des interactions. Par exemple, ils peuvent être utilisés pour gérer la publication de contenu, garantir la confidentialité, ou encore récompenser les utilisateurs pour leurs interactions de manière automatisée. Cette approche peut répondre à certaines problématiques des réseaux sociaux traditionnels, notamment en matière de contrôle des données et de monétisation des interactions.

Dans cet atelier, nous allons simuler des opérations simples de gestion de posts sur les réseaux sociaux en utilisant des smart contracts. Nous utiliserons **Remix IDE**, un

environnement de développement pour **Solidity**, et **MetaMask**, un portefeuille de crypto-monnaies, pour faciliter les interactions avec la blockchain. Cette simulation permettra de mieux comprendre le potentiel des smart contracts pour les réseaux sociaux, tout en découvrant les bases de la programmation blockchain.

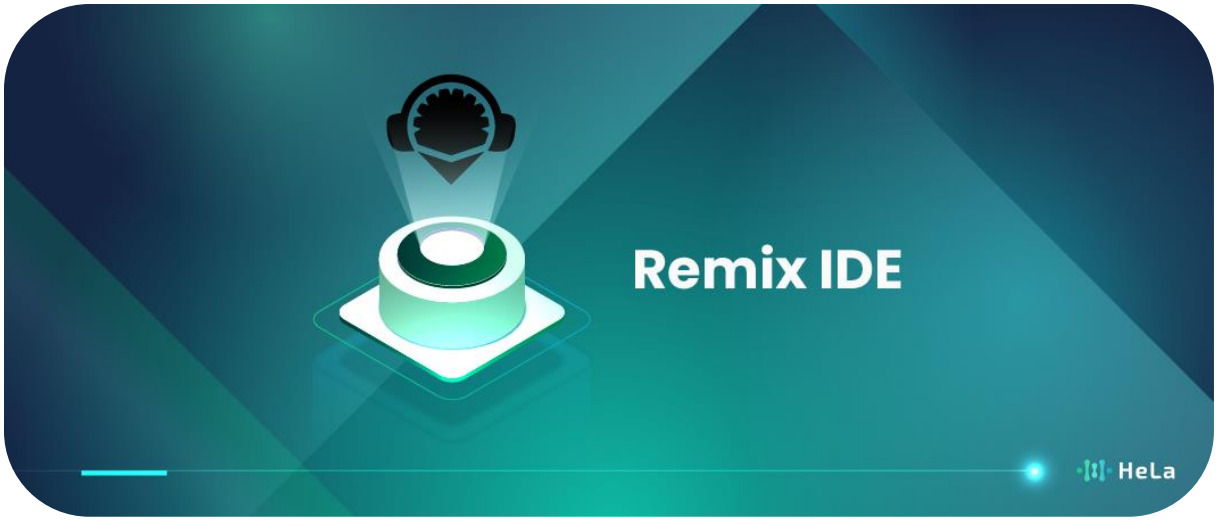
2. Solidity :

Solidity est un langage de programmation orienté objet, similaire à JavaScript, Python ou C++. Il est conçu pour développer des smart contracts sur la blockchain Ethereum, ainsi que sur d'autres blockchains compatibles, comme Tomochain. Pour faciliter son adoption par les développeurs et permettre aux débutants de l'apprendre aisément, Solidity s'inspire de nombreux autres langages, notamment JavaScript, C++ et C#. À l'origine, le projet a été proposé par Gavin Wood, cofondateur et ancien CTO d'Ethereum, et son développement a débuté en 2014 par l'équipe d'Ethereum. Solidity fait partie des quatre langages de programmation qui permettent d'interagir avec l'EVM (Ethereum Virtual Machine). Il est couramment utilisé pour développer des dApps et a également été impliqué dans d'autres projets, tels qu'un Proof of Concept (PoC) de SWIFT sur une blockchain privée, Borrow. Techniquement, Solidity est un langage orienté objet, un paradigme de programmation qui repose sur l'interaction entre des objets, qui sont des instances de concepts ou d'entités. Comme de nombreux autres langages, il prend en charge la manipulation de fonctions, de structures et de chaînes de caractères (strings).

3. Remix IDE :

Remix IDE est un environnement de développement intégré (IDE) spécialement conçu pour écrire, déboguer et déployer des smart contracts en Solidity sur la blockchain Ethereum. Accessible directement via un navigateur web, Remix offre une interface conviviale qui permet aux développeurs de coder et tester leurs smart contracts sans nécessiter de configuration complexe. Voici les Caractéristiques principales de Remix IDE :

- Une **interface** dans laquelle nous pouvons développer en Solidity.
- Un **compilateur** de notre choix pour nos smart-contracts.
- Un outil de récupération du code compilé des contrats pour les déployer.
- Un **débogueur** inclus pour analyser et corriger notre code.
- Une **fonction d'analyse** des transactions, des frais de gas etc.
- Un **mode nuit**.
- Un **outil de déploiement** sur une blockchain utilisant différents moyens.
- Une utilisation simplifiée des smart-contracts **OpenZeppelin**.
- Tout ça réuni sur une seule et même interface Web.



4. Metamask :

MetaMask est un portefeuille numérique et une extension de navigateur qui facilite l'accès aux applications décentralisées (dApps) et la gestion des actifs numériques sur la blockchain Ethereum et d'autres réseaux compatibles. Il sert de pont entre un navigateur web classique et le monde de la blockchain, permettant aux utilisateurs d'interagir facilement avec des contrats intelligents, de gérer des cryptomonnaies et d'accéder à des dApps directement depuis leur navigateur ou leur application mobile. MetaMask propose des fonctionnalités de gestion de portefeuille sécurisées pour stocker, envoyer et recevoir des cryptomonnaies, notamment l'Ether (ETH) et les tokens ERC-20. Il permet également une connexion rapide et sécurisée à une variété de dApps, telles que les plateformes de finance décentralisée (DeFi), les jeux blockchain et les places de marché de NFT. En plus d'Ethereum, MetaMask prend en charge d'autres réseaux comme Binance Smart Chain et Polygon, et permet aux utilisateurs de basculer facilement entre les réseaux ou d'ajouter de nouveaux réseaux personnalisés. Côté sécurité, MetaMask chiffre les clés privées et les stocke localement, et chaque transaction ou connexion aux dApps nécessite une confirmation manuelle de l'utilisateur. Disponible en tant qu'extension pour navigateurs (Chrome, Firefox, Edge, Brave) et en application mobile (iOS et Android), MetaMask est conçu pour rendre l'interaction avec la blockchain accessible tant aux

débutants qu'aux utilisateurs expérimentés, et reste particulièrement prisé dans les communautés DeFi et NFT pour sa simplicité et ses nombreuses options d'intégration.

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.

```
13    //Question3: Création de la fonction de publication.
14    function publishPost(string memory _message) public{    infinite gas
15        Post memory new_post = Post({
16            message: _message,
17            author: msg.sender
18    });
19    posts.push(new_post);
20    }
```

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

```
22    //Question4: Création de function de consultation.
23    function getPost(uint index) public view returns (string memory, address){    infinite gas
24        Post storage post =posts[index];
25        return (post.message,post.author);
26    }
27
```

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

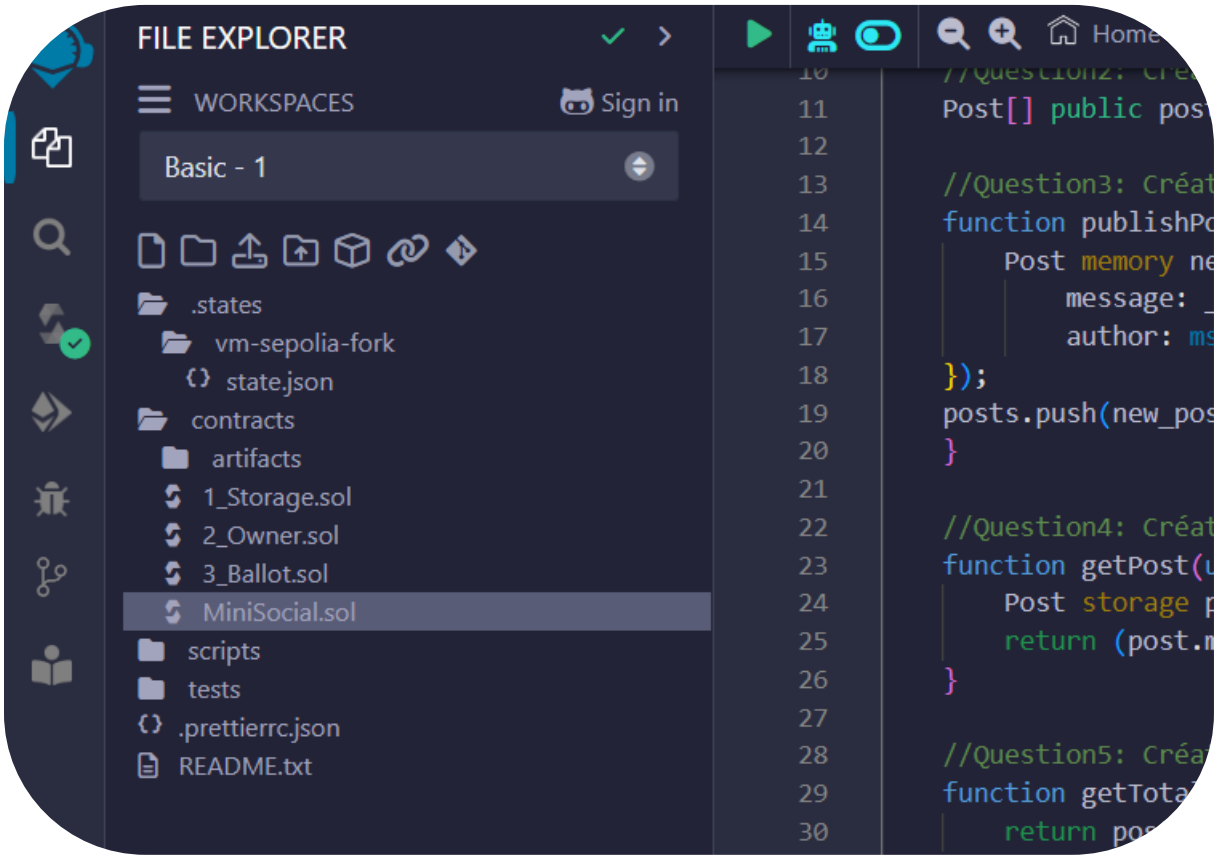
```
28    //Question5: Création de fonction pour récupérer le nombre des messages publiés.
29    function getTotalPosts() public view returns (uint){    2462 gas
30        return posts.length;
31    }
```

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 :

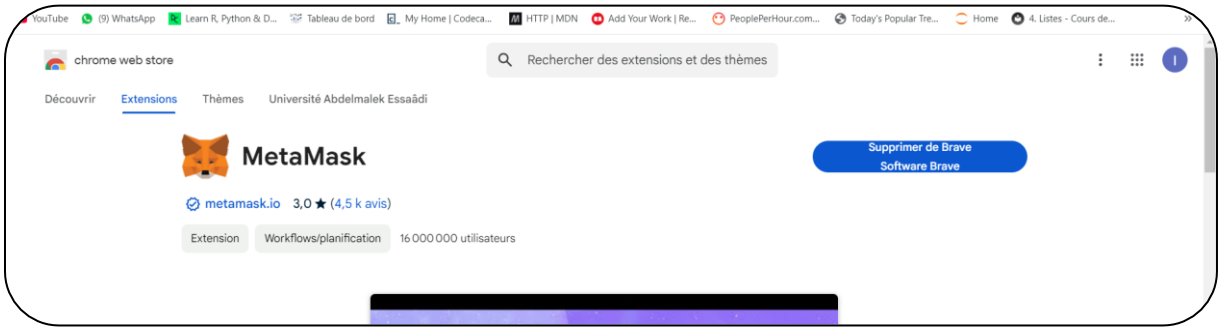
Étape 1 : Déclaration du Smart Contract :

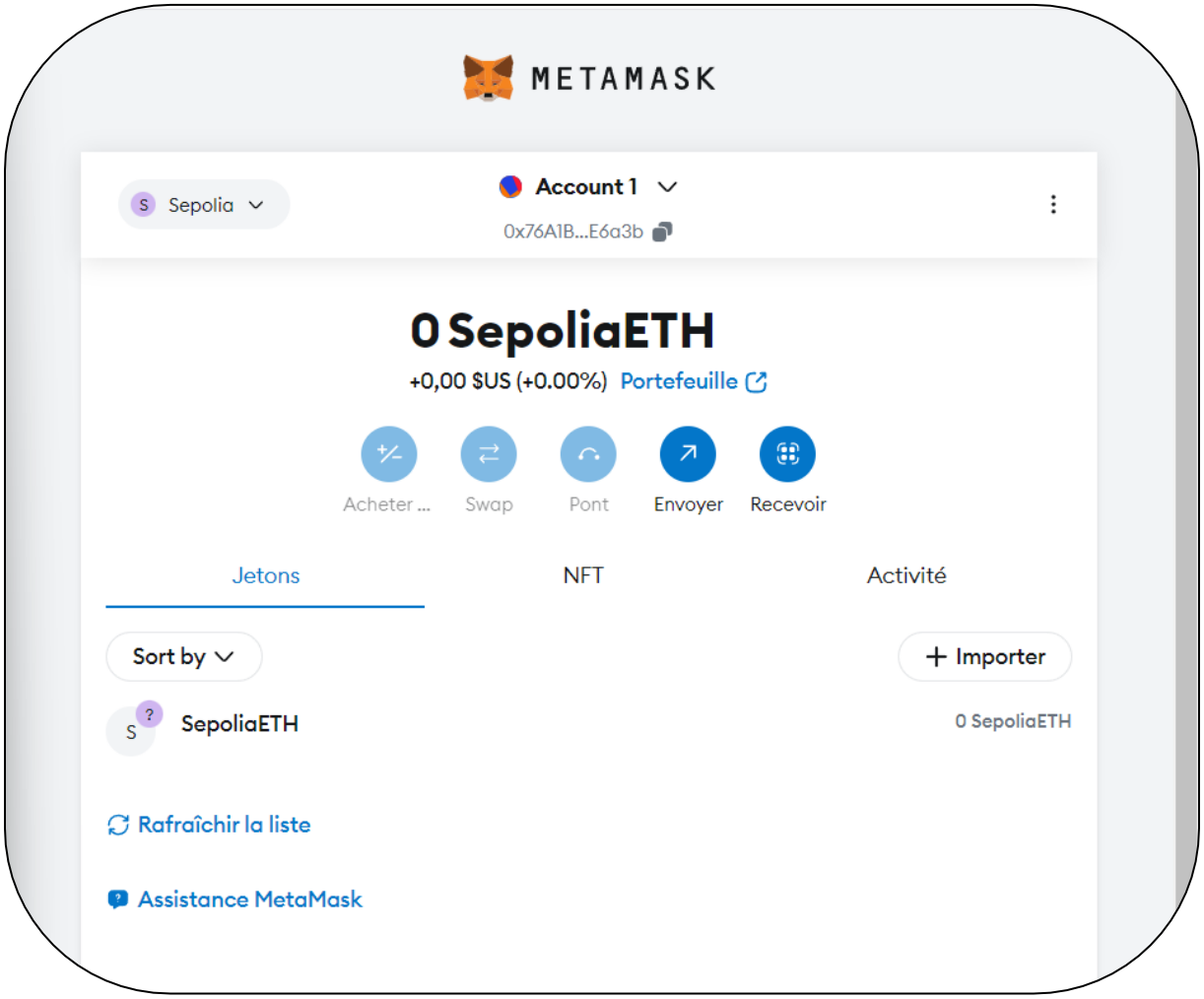
- Ouvrir REMIX IDE dans un navigateur (ce projet compte sur le navigateur BRAVE, https://remix.ethereum.org).
- Créer un fichier MiniSocial.sol
- Écrire le code de la smart contrat avec Solidity dans un fichier *MiniSocial.sol*



Étape 2 : Déploiement et Tests :

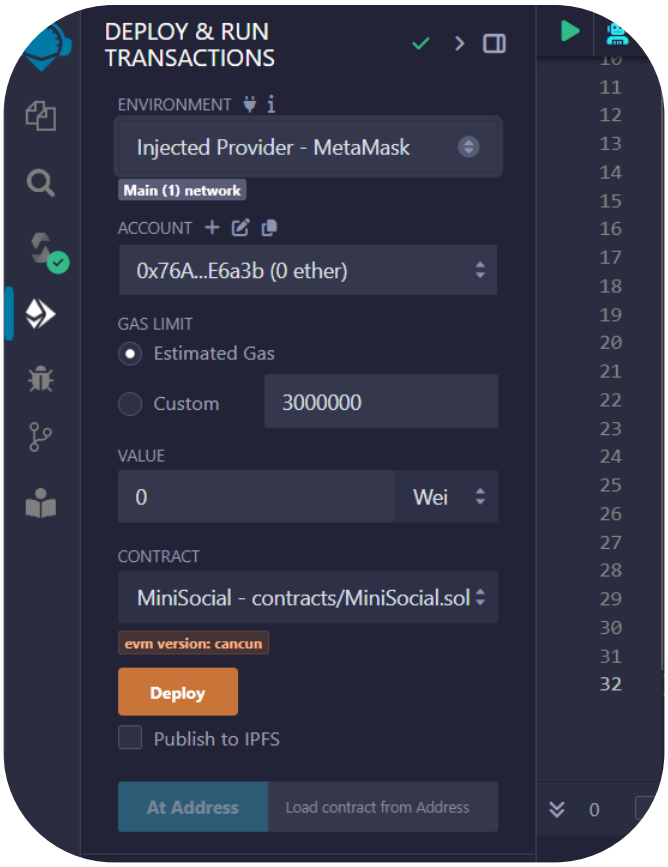
- **Installer l'extension MetaMask** : Assurez que l'extension MetaMask est installée et configurée sur le réseau de test Sepolia.



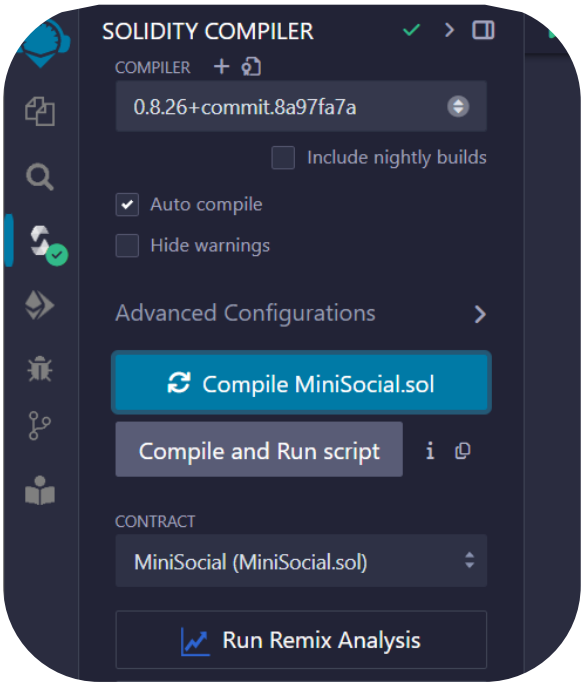


- **Déploiement du contrat**

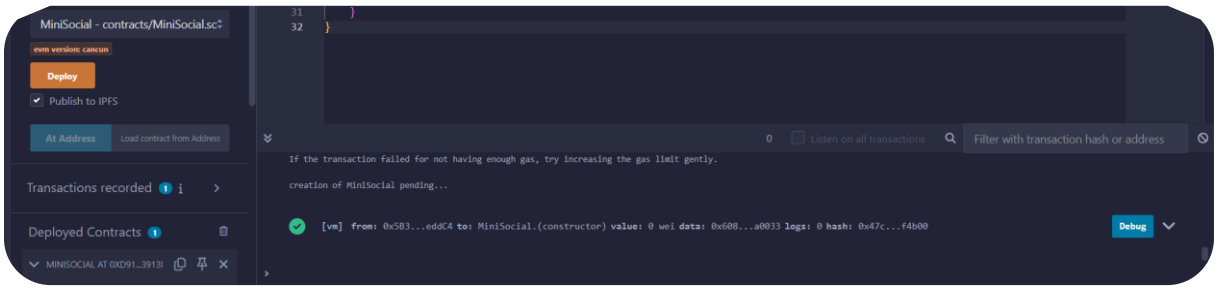
Dans Remix, connectez MetaMask en choisissant le réseau Sepolia dans les paramètres de Remix.

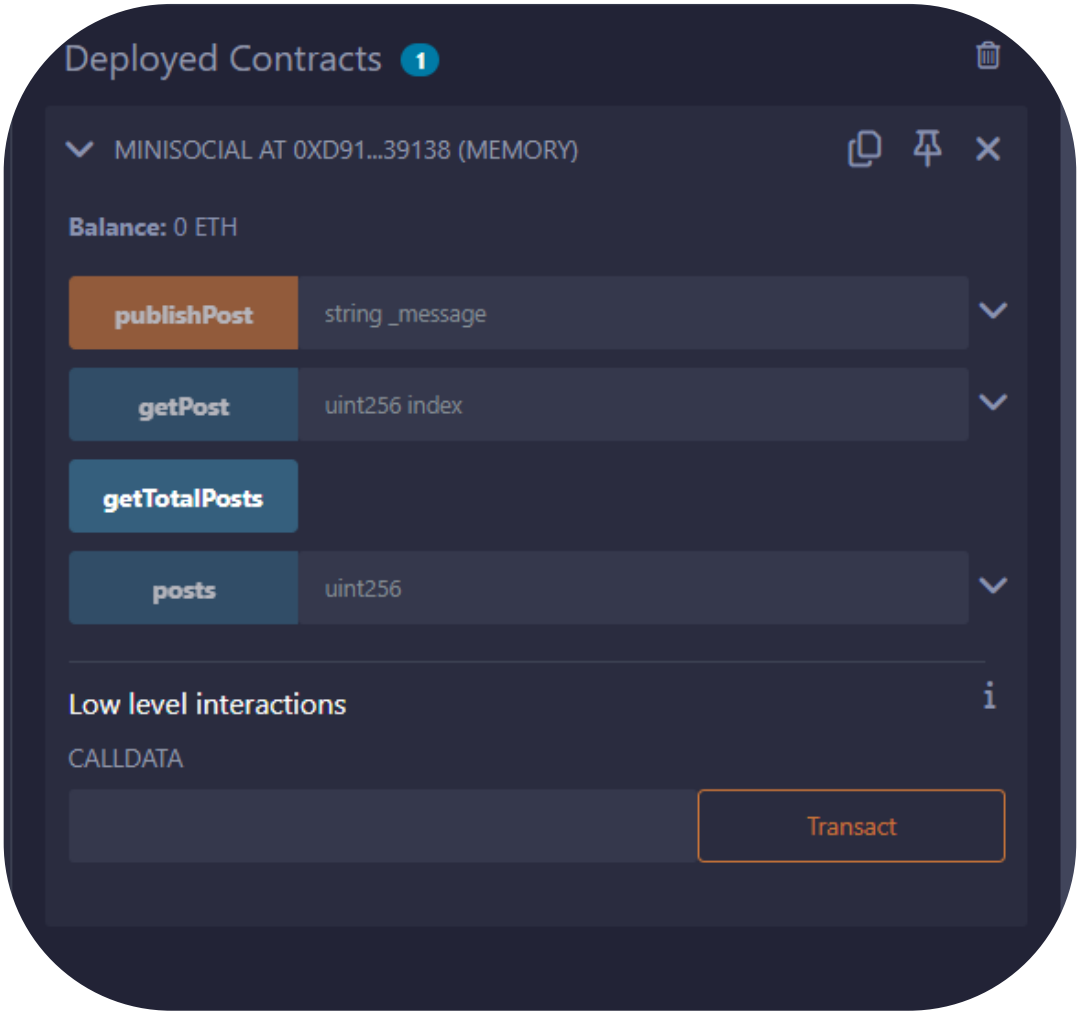


Compilez le contrat (MiniSocial.sol) en sélectionnant la bonne version de Solidity.



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

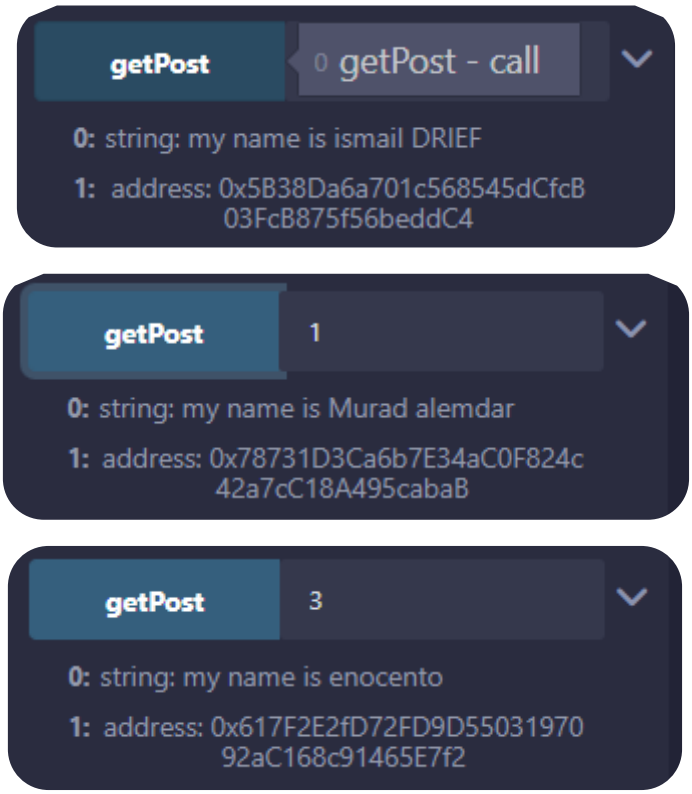




- **Test de la fonction publishPost** :

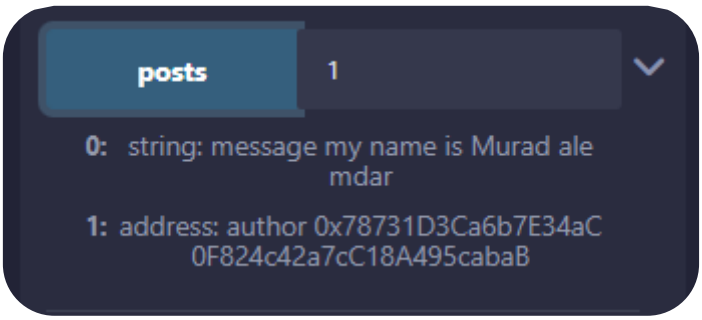
Utilisez plusieurs comptes pour appeler publishPost avec différents messages. Chaque appel enregistre un message et l'adresse de l'utilisateur dans le tableau posts.

- **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, consistant en la création d'un mini réseau social décentralisé, a permis d'explorer les fonctionnalités fondamentales des smart contracts en Solidity. Nous avons conçu et implémenté le contrat intelligent MiniSocial, intégrant des fonctionnalités essentielles telles que la publication et la consultation de messages ainsi que le suivi du nombre total de postes. Ce processus a été facilité par l'utilisation de Remix IDE pour le déploiement et les tests, ainsi que de MetaMask pour simuler des transactions sur un réseau de test. Ce projet met en avant l'importance des smart contracts pour la décentralisation des réseaux sociaux, offrant transparence et immuabilité aux interactions des utilisateurs. Les smart contracts garantissent que chaque message est associé à un auteur unique et stocké de manière sécurisée et accessible publiquement, sans dépendance à une autorité centrale. Cela ouvre des perspectives prometteuses pour les applications décentralisées, permettant aux utilisateurs de contrôler leurs données et leurs interactions dans un environnement sûr et fiable. En conclusion, ce projet a démontré la puissance des smart contracts pour créer des systèmes transparents et décentralisés, offrant une base solide pour des applications autonomes et résistantes à la censure, tout en posant les fondements d'une nouvelle ère pour les réseaux sociaux et les services numériques.