جامعة عبد المالك السعدي
Université Abdelmalek Essaâdi

كلية العلوم و التقنيات بطنجة
Faculté des Sciences et Techniques de Tanger

ROYAUME DU MAROC
UNIVERSITÉ ABDELMALEK ESSAADI
FACULTÉ DES SCIENCES ET TECHNIQUES - TANGER

CYCLE MASTER EN SCIENCES ET TECHNIQUES

FILIÈRE : INTELLIGENCE ARTIFICIELLE ET SCIENCES DE DONNÉES

BLOCKCHAIN
(ATELIER - 5)

RÉALISÉE PAR :
NOUIH Omar

ENCADRÉ PAR :
Pr. BENADBELOUAHAB Ikram

IASD
MASTER INTELLIGENCE ARTIFICIELLE ET SCIENCES DE DONNÉES

Année Universitaire 2024 - 2025

Smart Contract MiniSocial

Déclaration du Smart Contract

Le smart contract `MiniSocial` est implémenté en Solidity. Ce contrat permet aux utilisateurs de publier des messages sous forme de publications sur une plateforme sociale, où chaque message est associé à un auteur identifié par son adresse Ethereum. L'événement `NewPostEvent` est défini pour signaler la création d'une nouvelle publication.

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

contract MiniSocial {
    event NewPostEvent(uint indexed postId,
                       string message,
                       address indexed author,
                       uint timestamp);
}
```

Structure de Données : Commentaire et Publication

Le contrat contient deux structures de données :

- **Comment** : contient l'auteur du commentaire, le contenu et le nombre de likes du commentaire.
- **Post** : contient l'identifiant de la publication, le message, l'auteur, le nombre de likes et un tableau dynamique de commentaires associés.

```
struct Comment {
    address author;
    string content;
    uint numLikes;
}

struct Post {
    uint id;
    string message;
    address author;
    uint numLikes;
    Comment[] comments;
}
```

Stockage des Publications

Un tableau dynamique `posts` de type `Post[]` est déclaré pour stocker toutes les publications. Une structure de mapping `hasLiked` est utilisée pour gérer les likes de chaque utilisateur sur chaque publication, garantissant qu'un utilisateur ne puisse liker une publication qu'une seule fois.

```
Post[] private posts;
mapping(uint => mapping(address => bool)) private hasLiked;
```

Fonction de Publication

La fonction `publishPost(string memory _message)` permet à un utilisateur de publier un message d'une longueur maximale de 140 caractères. Si le message est valide, il est ajouté au tableau des publications avec l'auteur et l'horodatage.

```
function publishPost(string memory _message) public {
    require(bytes(_message).length > 0, "Post cannot be empty.");
    require(bytes(_message).length <= 140, "Post is too long.");

    uint postId = posts.length;
    Post storage newPost = posts.push();
    newPost.id = postId;
    newPost.message = _message;
    newPost.author = msg.sender;
    newPost.numLikes = 0;

    emit NewPostEvent(postId, _message, msg.sender, block.timestamp);
}
```

Consultation des Auteurs des Publications

La fonction `getPostAuthor` renvoie l'adresse de l'auteur d'une publication donnée, en vérifiant que l'identifiant de la publication est valide.

```
function getPostAuthor(uint _postId) public view returns (address) {
    require(_postId < posts.length, "Invalid post ID.");
    return posts[_postId].author;
}
```

Consultation des Likes des Publications

La fonction `getPostLikes` renvoie le nombre de likes d'une publication spécifique.

```
function getPostLikes(uint _postId) public view returns (uint) {
    require(_postId < posts.length, "Invalid post ID.");
    return posts[_postId].numLikes;
}
```

Nombre Total de Publications

La fonction `getTotalPosts()` renvoie le nombre total de publications sur la plateforme.

```
function getTotalPosts() public view returns (uint) {
    return posts.length;
}
```

Consultation d'une Publication

La fonction `getPost` renvoie le contenu et l'auteur d'une publication en fonction de son identifiant.

```
function getPost(uint _postId) public view returns (string memory, address) {
    require(_postId < posts.length, "Invalid post ID.");
    Post storage post = posts[_postId];
    return (post.message, post.author);
}
```

Ajout de Likes aux Publications

La fonction `addLike` permet aux utilisateurs d'ajouter un like à une publication. Chaque utilisateur ne peut liker une publication qu'une seule fois grâce au mapping `hasLiked`.

```
function addLike(uint _postId) public {
    require(_postId < posts.length, "Invalid post ID.");
    require(!hasLiked[_postId][msg.sender], "You have already liked this post.");

    posts[_postId].numLikes++;
    hasLiked[_postId][msg.sender] = true;
}
```

Ajout de Commentaires

La fonction `addComment` permet à un utilisateur d'ajouter un commentaire à une publication. Le commentaire est associé à l'auteur et au contenu.

```
function addComment(uint _postId, string memory _content) public {
    require(_postId < posts.length, "Invalid post ID.");
    require(bytes(_content).length > 0, "Comment cannot be empty.");

    Post storage post = posts[_postId];
    post.comments.push(Comment({
        author: msg.sender,
        content: _content,
        numLikes: 0
    }));
}
```

Consultation des Commentaires d'une Publication

La fonction `getPostComments` permet de récupérer tous les commentaires associés à une publication spécifique.

```
function getPostComments(uint _postId) public view returns (Comment[] memory) {
    require(_postId < posts.length, "Invalid post ID.");
    return posts[_postId].comments;
}
```

Ajout de Likes aux Commentaires

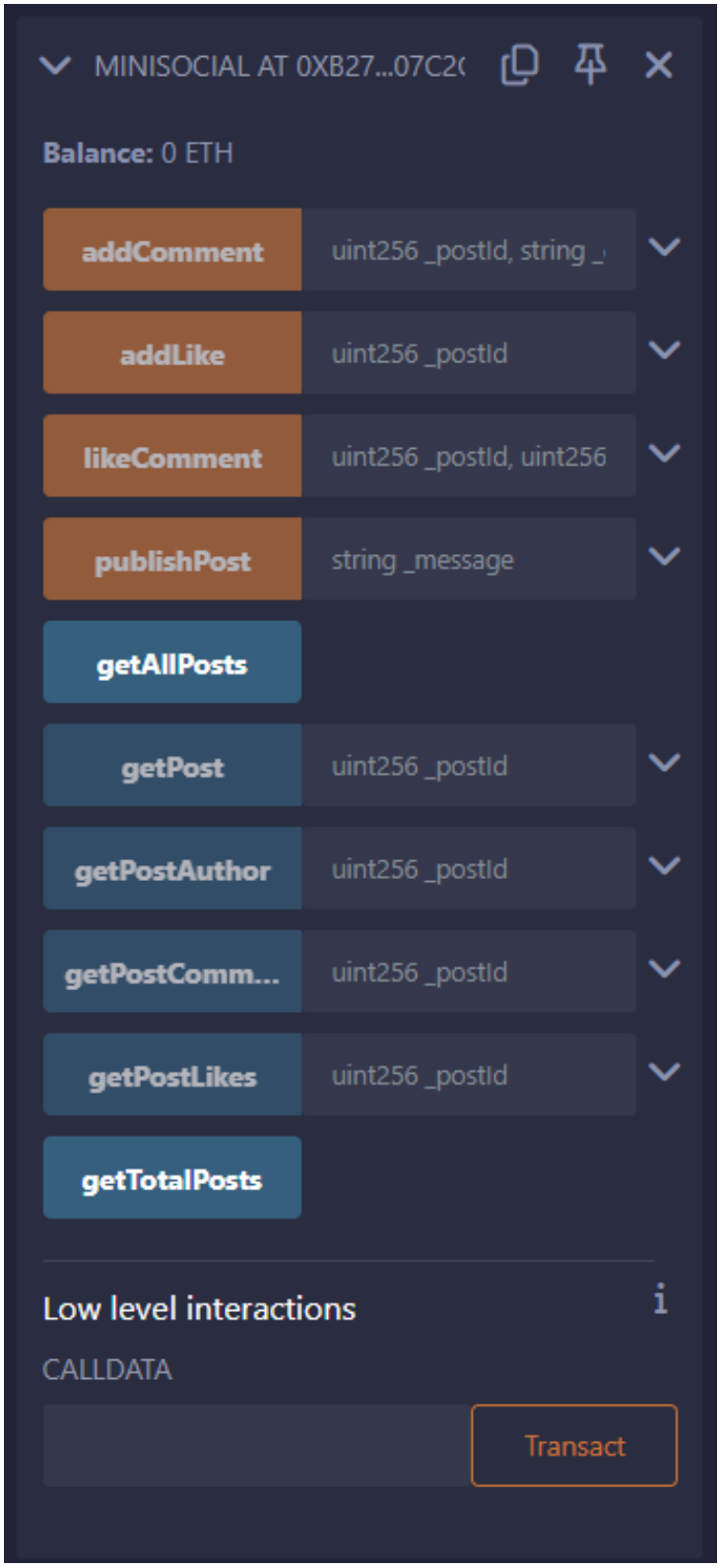
La fonction `likeComment` permet aux utilisateurs d'ajouter un like à un commentaire spécifique dans une publication.

```
function likeComment(uint _postId, uint _commentId) public {
    require(_postId < posts.length, "Invalid post ID.");
    Post storage post = posts[_postId];
    require(_commentId < post.comments.length, "Invalid comment ID.");

    post.comments[_commentId].numLikes++;
}
```

Déploiement du Smart Contract

Pour déployer le smart contract `MiniSocial`, j'ai d'abord essayé d'utiliser MetaMask, mais j'ai rencontré un problème de solde insuffisant pour payer les frais de transaction en Ether (ETH). Finalement, j'ai décidé de déployer le contrat sur une machine virtuelle (VM) où je pouvais exécuter le contrat sans ces contraintes.



Explication de l'image

L'image montre les fonctions du contrat `MiniSocial`, telles que `addComment`, `addLike`, et `publishPost`. Ces fonctions permettent aux utilisateurs de publier des messages, d'ajouter des likes, et de commenter. Bien que MetaMask n'ait pas été utilisé pour le déploiement final, ces fonctionnalités sont toujours accessibles via la VM.

Étapes de Déploiement sur la VM

Voici comment j'ai déployé le contrat sur la VM :

1. J'ai importé le code du contrat dans un environnement Solidity sur la VM.
2. J'ai exécuté le contrat directement sur la VM, ce qui m'a permis de le tester sans avoir besoin d'ETH.

Cette solution m'a permis de surmonter le problème de solde et de tester le contrat de manière efficace.

Conclusion

Le projet `MiniSocial` montre comment créer un réseau social décentralisé. Le déploiement sur une VM a permis d'éviter les frais de transaction associés aux réseaux blockchain publics comme Ethereum.