Report: Smart Contract Implementation and Testing for MiniSocial

Made by : RIYADI ADNANE

Requested Implementation:

1. Declaration of the Smart Contract and Basic Structure

The smart contract was declared under the name MiniSocial, providing a central framework for managing user-generated content. Within this contract, we introduced a structure called `Post` to hold essential data for each message. Each `Post` includes:

- `message`: a string representing the content of the user's message.
- `author`: the Ethereum address of the user who published the message. This setup allows each post to be securely associated with a unique message and its author, ensuring that contributions to the social feed are stored in a decentralized, tamper-proof way.

2. Message Storage Using a Dynamic Array

To store the user posts, we defined a dynamic array named `posts`, of type `Post[]`. This array acts as the main data store for all published messages, allowing us to add and organize content efficiently. By utilizing a dynamic array, we ensure that each post can be easily indexed and retrieved as needed, which simplifies data access for both users and the contract itself.

3. Function to Publish a Message

The `publishPost` function allows users to publish their messages on the blockchain. This function takes one parameter, `_message`, which is the content of the post. When a user calls `publishPost`, the function:

    - Creates a new `Post` object containing the message and the sender's Ethereum address (`msg.sender`).

- Appends this new post to the `posts` array. By storing the sender's address along with each message, we create a permanent, immutable record of authorship, ensuring that each post is correctly attributed to its creator.

4. FunctionFunction to Retrieve a Post by Index and Get the Total Number of Posts

To allow users to access specific posts, we implemented the `getPost` function, which accepts an index as its parameter. This function retrieves and returns both the message text and the author's address for the specified post. Additionally, the `getTotalPosts` function returns the length of the `posts` array, which represents the total number of posts made. This information allows users to easily determine the range of available posts and assess the volume of content shared on the MiniSocial contract.

Improved Implementation:

1. Additional Features in Post Structure

In the initial code, each post contained only a `message` and `author` field. The improved version introduces two new fields, `likes` and `timestamp`, in the `Post` structure. The `likes` field tracks user engagement by counting the number of times a post has been liked, while the `timestamp` field stores the post's creation time. This timestamp allows for time-limited interactions, specifically within a 24-hour window, which aligns with common social media “story” functionalities, adding a dynamic aspect to user engagement.

2. Comments Feature

The improved contract includes a new `Comment` structure with `text`, `commenter`, and `timestamp` fields, allowing users to add comments to posts. These comments are stored in a mapping (`postComments`) that associates each post with an array of comments. This addition introduces a more comprehensive social interaction layer, where users can respond to posts, fostering conversations around content.

3. Like Feature with User Constraints

The new version enables users to like posts, providing a more interactive experience. The contract includes a `likePost` function, which increments the `likes` count on a post. A mapping, `hasLiked`, prevents users from liking the same post multiple times. Furthermore, the function includes a 24-hour expiration condition, requiring that posts be liked within this timeframe. This limitation not only prevents double-liking but also aligns with the social media trend of ephemeral content, encouraging timely engagement.

4. Enhanced Post Retrieval with Expiration Check

The `getPost` function in the updated contract now checks if the post is still within the 24-hour active window. If the post has expired, the function returns empty values, signaling that the post is no longer viewable. This change improves the UX by ensuring that only relevant and active content remains accessible, reflecting real-time engagement patterns.

5. Comment Count and Retrieval

In addition to the comments feature, the updated contract offers a `getCommentCount` function to retrieve the number of comments associated with a post. The function `getComment` further allows users to access specific comments based on their index. This level of access and data structuring facilitates detailed post interactions, enabling users to view both the quantity and specific content of comments on a post.

6. Comprehensive Post Details Function

A new `getPostDetails` function consolidates post and comment data for a more informative content retrieval experience. This function returns not only the post's `message`, `author`, and `likes`, but also the count and details of all comments. By integrating both text and author of each comment, it provides a full overview of each post and the associated interactions, reflecting a complete and engaging social media experience.

7. Total Post Count (Active Posts)

The `getTotalPosts` function remains similar to the initial implementation but could be extended in future iterations to return the count of active posts only, based on their expiration timestamps. This approach could streamline content retrieval by focusing on currently relevant posts.

Testing and Deployment Results

The updated `MiniSocial` contract was successfully tested and deployed on a virtual machine (VM), confirming that all new functionalities, including likes, comments, and post expiration logic, work as expected. The contract operated smoothly in a controlled VM environment, validating each new feature and ensuring the contract's reliability. However, when deploying to the MetaMask test network, issues were encountered that prevented full functionality.

Conclusion

The improved `MiniSocial` contract introduces substantial enhancements in interactivity, functionality, and user engagement. These changes, including likes, comments, and time-limited posts, align the contract with social media trends and create a more dynamic,

community-driven environment for users. The additions represent a shift from a simple post management system to a more interactive and feature-rich social contract.

U