SOFTWARE REQUIREMENTS SPECIFICATION

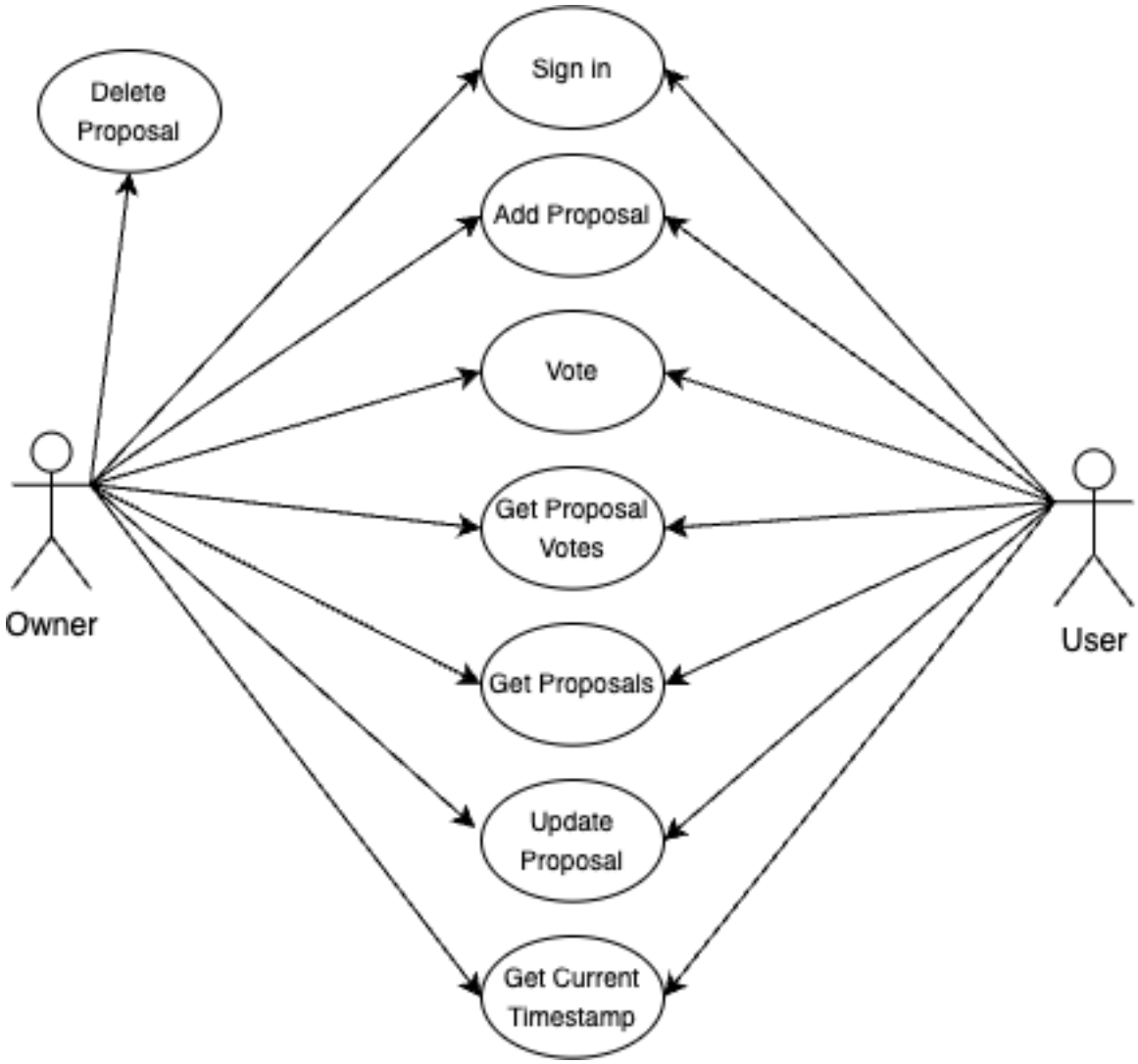
FUNIX VOTING DAAP

I. Product overview

1. **`Product name:`** Funix Voting Daap
2. **`Product description:`** The Voting App is a decentralized application (DApp) designed to facilitate transparent and secure voting processes on the Ethereum blockchain. It allows users to propose, vote on, and manage various proposals in a decentralized and trustless environment.
3. **`Voters:`** Individuals or entities who participate in the voting process by creating proposals, casting votes.
4. **`Admins:`** Administrators responsible for managing the voting process, overseeing proposals, and delete proposals when needed.
5. **`The benefit to users:`**
6. **`Participation in Decision-making:`** The Voting App empowers users to have a direct say in decision-making processes by allowing them to propose new ideas and vote on existing proposals. This active participation fosters a sense of ownership and involvement within the community.
7. **`Transparency and Trust:`** By leveraging blockchain technology, the Voting App ensures transparency and trust in the voting process. Users can verify the integrity of the voting outcomes and trust that their votes are accurately counted and recorded on the immutable Ethereum blockchain.
8. **`Community Engagement:`** Engaging in the voting process through the Voting App fosters community engagement and collaboration. Users can contribute their ideas, express their opinions, and collectively shape the direction of the community or organization.
9. **`Efficiency and Accessibility:`** The Voting App streamlines the voting process, making it more efficient and accessible to users. With just a few clicks, users can create proposals, cast their votes, and stay updated on the status of ongoing voting activities.

10. **`Problem-solving and Decision-making:`** The Voting App provides a platform for users to address issues, propose solutions, and make informed decisions collectively. By aggregating the collective wisdom of the community, the app facilitates problem-solving and effective decision-making processes.

`11. Functional Diagram:`



II. Functional Requirements

1. **Function:** addProposal
    a. **Description:** This function allows users to add a new proposal with a specified title and description. It creates a new proposal object and adds it to the list of existing proposals.
    b. **Users to Use:** All users and admins.
    c. **Function Inputs:**
        i. _title: Title of the proposal (string).
        ii. _description: Description of the proposal (string).
    d. **Rule:** Both title and description must not be empty.
    e. **Results:** Upon successful execution, a new proposal is created and added to the list of proposals. The proposal ID is incremented, and its details including title, description, and initial vote counts are stored in the blockchain.
2. **Function:** vote
    a. **Description:** This function allows users to cast their votes (either yes or no) for a specific proposal identified by its proposal ID.
    b. **Users to Use:** All users and admins.
    c. **Function Inputs:**
        i. _proposalId: ID of the proposal to vote on (uint256).
        ii. _yesVote: Boolean indicating whether the user is voting "yes" (true) or "no" (false) for the proposal.
    d. **Rule:** The proposal ID must be valid. It should be less than the total number of proposals. Each user can vote only once for a specific proposal.
    e. **Results:** Upon successful execution, the user's vote is recorded for the specified proposal. The vote count for the proposal is updated accordingly, and the user is prevented from voting again for the same proposal.
3. **Function:** getProposalVotes
    a. **Description:** This function retrieves the number of "yes" and "no" votes for a specific proposal identified by its proposal ID.
    b. **Users to Use:** All users and admins.
    c. **Function Inputs:**

    i. _proposalId: ID of the proposal to retrieve votes for (uint256).

d. **Rule:** The proposal ID must be valid. It should be less than the total number of proposals.

e. **Results**: Upon successful execution, the function returns the number of "yes" and "no" votes for the specified proposal. This information can be used to display the voting results to users.

4. **Function:** getProposals
    a. **Description:** This function retrieves all non-deleted proposals stored in the contract.
    b. **Users to Use:** All users and admins.
    c. **Function Inputs:** None.
    d. **Rule:** None.
    e. **Results:** Upon successful execution, the function returns an array of Proposal objects representing all non-deleted proposals stored in the contract. Each proposal object contains details such as ID, title, description, vote counts, and deletion status. This information can be used to display the list of proposals to users.

5. **Function:** deleteProposal
    a. **Description:** This function allows the owner of the contract to delete a proposal by marking it as deleted.
    b. **Users to Use:** Admins or owner of the contract.
    c. **Function Inputs:**
        i. _proposalId: ID of the proposal to be deleted (uint256).
    d. **Rule:** The caller of this function must be the owner of the contract.
    e. **Results:** Upon successful execution, the function marks the specified proposal as deleted by setting its isDeleted flag to true. Additionally, it removes the proposal from the array of proposals. This prevents the proposal from being displayed in the list of proposals retrieved by other functions such as getProposals().

6. **Function:** updateProposal

a. **Description:** This function allows the owner to update the details of a proposal. It can modify the title, description, and end time of a proposal.
b. **Users to Use:** Admins or owner of the contract.
c. **Function Inputs:**
    i. _proposalId (uint256): ID of the proposal to update.
    ii. _title (string): New title of the proposal.
    iii. _description (string): New description of the proposal.
    iv. _endsAt (uint256): New time in hours when the proposal ends.
d. **Rule:** The proposal ID must be valid. The proposal must not be deleted or already have votes. The caller must be the owner of the contract.
e. **Results:** Upon successful execution, the specified proposal's details are updated with the new title, description, and end time.

7. **Function:** getCurrentTimestamp
    a. **Description:** This function retrieves the current timestamp of the blockchain.
    b. **Users to Use:** All users and admins.
    c. **Function Inputs:** None.
    d. **Rule:** None.
    e. **Results:** Upon successful execution, the function returns the current timestamp as a uint256. This information can be used to display the current time or compare with proposal timings.

III. Non-Functional Requirements

1. **Web Browser Coverage:**
    a. The product should support the following popular web browsers:
        i. Google Chrome
        ii. Mozilla Firefox
        iii. Safari
        iv. Microsoft Edge

2. **Mobile Compatibility:** The product should be accessible and provide a seamless user experience on mobile devices, including smartphones and tablets.
3. **Transaction Processing Time:**
    a. Each transaction should be processed within a reasonable time frame, typically a few seconds.
4. **Technical Requirements for Interaction:**
    a. Provider
        i. Users must have the Metamask browser extension installed to interact with the application.
        ii. Users must add the Optimism Sepolia testnet to their Metamask accounts to access the application's functionality in the testnet environment.

IV. Product Upgrades

1. **Proposal Categories:**
    a. Add an option for users to suggest new categories for proposals. Implement a moderation system where admins can approve or reject suggested categories.
    b. Allow users to filter proposals by multiple categories simultaneously to refine their search results further.
2. **User Interface (UI) Improvements:**
    a. Incorporate responsive design principles to ensure the dApp is accessible and user-friendly across various devices, including desktops, tablets, and smartphones.
    b. Enhance visual feedback for user actions, such as highlighting selected proposals, displaying loading indicators during data retrieval, and providing success/error messages after completing actions like voting or commenting.
3. **User Authentication:**
    a. Enhance security by implementing two-factor authentication (2FA) options, such as SMS or email verification, to add an extra layer of protection to user accounts.

b. Provide users with the ability to recover their accounts in case of forgotten passwords through a password reset functionality.

4. Proposal Expiry:

a. Introduce an option for proposal creators to extend the expiry period of their proposals if needed, allowing for flexibility based on proposal relevance and ongoing discussions.

b. Implement a notification system to alert users when a proposal they are following is nearing its expiry date, encouraging them to engage in voting or discussion before the deadline.