Task

Enhanced DAO Membership & Proposal System Specification

Objective: Create a DAO where membership is represented by an NFT. The DAO must support proposal creation, voting, and execution, while ensuring proper ownership management and governance.

Additionally, deploy your solution on Sepolia, and in your GitHub repository's README.md file, include the Sepolia address for deploying the contract. Mint the first NFT to an EOA, upload the metadata of the minted NFTs to IPFS, and include the corresponding IPFS hashes of these metadata files in the README.md.

Please ensure that the smart contract address you provide matches the smart contract code you have uploaded to GitHub.

Core Requirements:

1. **NFT Membership (ERC721)**
   - Purpose: The NFT serves as proof of DAO membership.
   - Initial Minting: Upon deployment, mint the first NFT to the DAO smart contract.
   - One NFT per Address: Enforce that each address can mint only one NFT.
   - Mint Price: 0.01 ETH per NFT.
   - Sequential Token IDs: NFTs should be minted sequentially starting from token ID 1.
   - Metadata Hosting: Host NFT metadata on IPFS with JSON fields: `name`, `description`, `image`. Ensure the metadata URI is set during minting.
2. **DAO Ownership & Proposal System**
   - Automatic Deployment & Minting: The DAO contract deploys the NFT contract, which mints the first NFT to the creator.
   - `createProposal` function: allowing only DAO members (NFT holders) to create proposals. The deadline for voting is 7 days from proposal creation.
   - Implement `voteForProposal`: allowing only DAO members to vote once per proposal. Voting options should be: For or Against.
   - Implement `executeProposal` to execute proposals only after the deadline has passed. A proposal is approved if *votesFor* > *votesAgainst*.

```
struct Proposal {
    address proposalCreator;
    string description;
    uint256 deadline;
    uint256 votesFor;
    uint256 votesAgainst;
    bool executed;
}
```

3. **Events & Custom Errors**
   - Define events such as *MembershipMinted*, *ProposalCreated*, *UserVoted*, *ProposalExecuted* for tracking success scenarios.
   - Define custom errors such as *NotNFTHolder*, *AlreadyVoted*, *ProposalNotFound*, *VotingPeriodEnded*, *ProposalAlreadyExecuted* for failure scenarios.

Additional Features (Optional but Recommended):

- Contract verification: Verify your smart contract on Etherscan.
- Support for ERC1155: Consider extending the membership system to ERC1155 tokens for multiple membership tiers.
- Enhanced Voting System: Use an `enum VotingChoice { For, Against, Abstain }` to determine proposal outcome based on: $votesFor > votesAgainst + votesAbstain$.
- Fund Management: Implement `Ownable` module logic to handle funds collected from NFT sales.
- Deterministic Deployment: Use the `create2` opcode for predictable contract addresses.
- Testing: Write comprehensive unit tests for each functionality using frameworks like Truffle, Hardhat, or Foundry.
- Upgradeability: Consider implementing proxy upgradeability patterns.