TP 5 - DecentraVote (180 minutes)

DecentraVote - A Democratic Governance System in Ancient Greece

Welcome to **Athens**, where democracy was born! The citizens of Athens gather to debate and vote on new laws, but the process needs an upgrade. With blockchain technology, we'll ensure fairness, transparency, and immutability in their democratic system.

Your task is to build a **voting dApp** that allows citizens to propose laws and vote on them, with all actions immutably recorded on the blockchain. Proposals will be finalized after **10 minutes** of voting to allow for quick testing during development.

Part 1: Workshop Breakdown

Guidance:

1. **The Problem**:
    - In ancient Athens, there was no way to ensure that every vote was counted fairly or that the proposed laws were stored safely.
    - Our blockchain-based system will solve this by:
        - Recording laws and votes immutably.
        - Allowing proposals and transparent voting.
2. **Plan**:
    - You will write a smart contract that handles proposals, voting, and finalization.
    - Then, following the guidelines, you will deploy it on a local blockchain for quick iteration and testing.

Part 2: Writing the Smart Contract (1 hour)

Guidance:

Open Remix IDE. Start with a minimal contract and progressively expand its functionality. Test each feature as you go.

1. **Structure**:
   Below are the function names. You will implement them step by step.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract AthenianDemocracy {
    struct Proposal {
        string ipfsHash;
        string description;
        address proposer;
        uint256 approvals;
        uint256 rejections;
        uint256 timestamp;
        bool finalized;
        bool passed;
    }
    mapping(uint256 => Proposal) public proposals;
    mapping(uint256 => mapping(address => bool)) public hasVoted;
    uint256 public proposalCount;
    function proposeLaw(string memory _description, string memory _ipfsHash) public { }
    function vote(uint256 _proposalId, bool _approve) public { }
    function finalizeProposal(uint256 _proposalId) public { }
}
```

2. **Steps**:

- **Step 1: Proposing a Law**
    - Write the `proposeLaw` function to add new laws with an IPFS hash and description.

    - **Guidance**:
        - Store the law's description and IPFS hash in the `Proposal` struct.
        - Record the proposer and the proposal's timestamp.
        - Increment `proposalCount`.
    - **Resource**: Solidity Mappings.
- **Step 2: Voting**
    - Implement the `vote` function. Citizens can cast votes (`true` for approval, `false` for rejection).
    - **Guidance**:
        - Ensure users cannot vote twice on the same proposal.
        - Track votes using `hasVoted`.
        - Increment `approvals` or `rejections` based on the vote.
    - **Resource**: Solidity Require Statement.
- **Step 3: Finalizing Proposals**
    - Implement the `finalizeProposal` function to determine whether a proposal is approved or rejected after 10 minutes.
    - **Guidance**:
        - Check if 10 minutes have passed.
        - Mark the proposal as `finalized` and update `passed` status.
    - **Resource**: Solidity Timestamp.

Challenge for Experts:

- Add a **quorum rule**: A proposal must receive votes from at least 20% of citizens to be valid.
- Add a **cool-down mechanism**: A proposer cannot submit two proposals within the same 24-hour period.

Part 3: Deploying and Testing (1 hour)

1. **Deploy**:

- Use Hardhat or Remix IDE to deploy the smart contract on a local blockchain.
- **Guidance**: Hardhat Getting Started.

2. **Test**:

- Submit a proposal and retrieve it.
- Cast votes and finalize the proposal after 10 minutes.
- Verify that the proposal status updates correctly.
- **Resource**: Testing with Hardhat.

Part 4: Building the Frontend (60-80 minutes)

Guidance:
Build a minimal React app to interact with the contract.

1. **Components (let the smart contract alone for now, only front-end with fake values)**:

- **Propose a Law**: Form to submit laws with description and IPFS hash.
- **Vote on Laws**: Display active proposals with `Approve` and `Reject` buttons.
- **Results**: Show finalized proposals with their status (approved/rejected).

2. **Integration**:

- Use either Ethers.js or Web3.js to connect the frontend to your smart contract.
- Integrate **Metamask authentication** for voter identity.
- **Resource**: Ethers.js Docs. Web3.js Docs.

Challenge for Experts:

- Add a **voter dashboard** showing a citizen's voting history and participation score.

Bonus: Integrating IPFS for Decentralized Storage (30 minutes)

- Use IPFS to store article content and retrieve the corresponding hashes.
- Write a Node.js script to pin content to IPFS using a public gateway (e.g., Infura or Pinata).

Challenges:

1. Implement file validation before pinning to IPFS (e.g., content length check).
2. Automate IPFS pinning and hash retrieval in the deployment script.

Part 5: Reflection and Discussion (15 minutes)

- **Debate**: How does decentralization improve democratic processes?
- **Discuss**: What are the limitations of on-chain voting for real-world use cases?

Activité précédente
Chapitre 05 : Coder une dApp

Aller à…

Activité suivante
Chapitre 06 : Introduction à la Sécurité des Smarts Contracts