

TRABALHO DE OFICINA DE DESENVOLVIMENTO DE SISTEMAS II

Guilherme Santos da Silva
Guilherme Lucas Teixeira Silva

Prof: Fabio Santos

INTRODUÇÃO

A tecnologia blockchain tem revolucionado a forma como lidamos com segurança, transparência e descentralização em sistemas digitais. Aplicar blockchain em um sistema de votação é uma alternativa promissora para evitar fraudes, garantir integridade dos votos e permitir auditoria pública. Este projeto propõe um sistema de votação descentralizado (dApp) na rede Ethereum usando contratos inteligentes.

PROBLEMA

Sistemas de votação tradicionais, principalmente digitais, enfrentam problemas como:

- Possibilidade de votos duplicados ou manipulados.
- Falta de transparência no processo.
- Falhas de segurança em bancos de dados centralizados.
- Dificuldade em garantir a confiança de todos os participantes no sistema.

OBJETIVO GERAL

Criar um sistema de votação que seja seguro, transparente, à prova de fraudes e acessível, utilizando blockchain e contratos inteligentes.

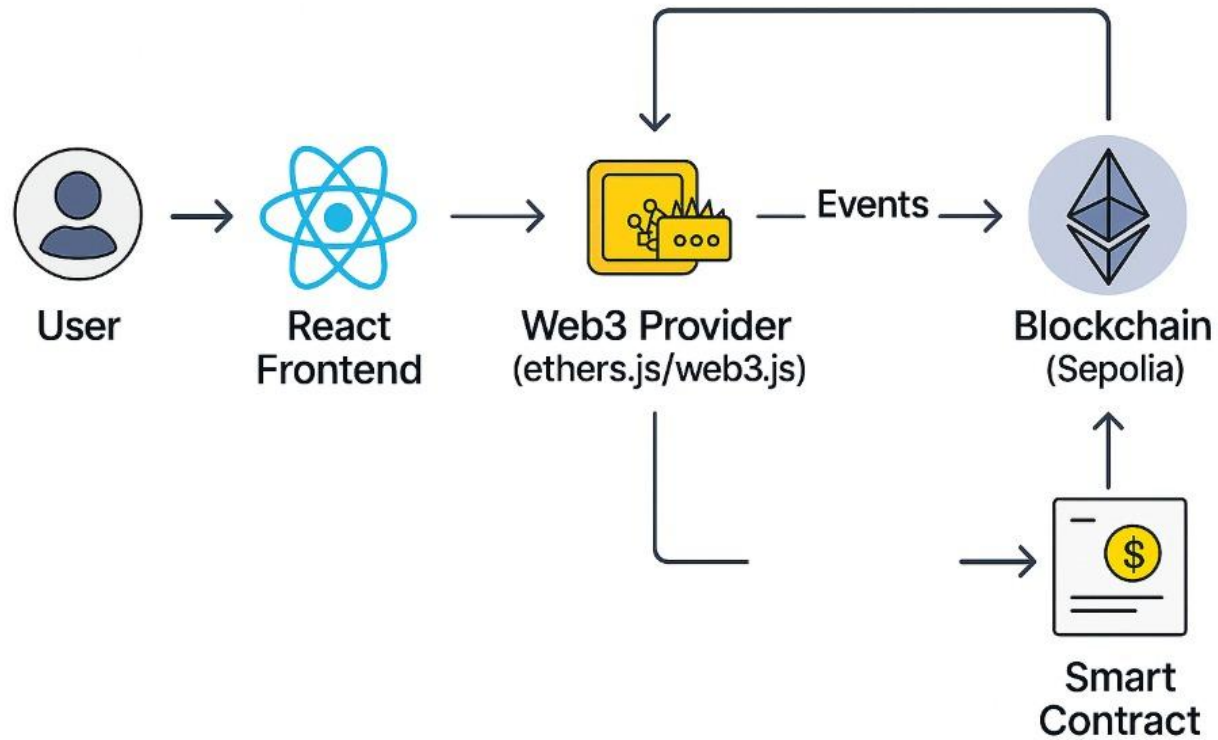
SOLUÇÃO PROPOSTA

Desenvolver um aplicativo descentralizado (dApp) composto por:

- Um contrato inteligente em Solidity, implantado na testnet Sepolia da rede Ethereum, responsável por gerenciar os votos, verificar se o eleitor já votou e contabilizar os resultados.
- Um frontend simples baseado em HTML, JavaScript e Web3.js, que interage com o contrato e permite que os usuários votem usando suas carteiras MetaMask.

Com isso, cada voto será imutável e público na blockchain, e cada usuário poderá votar apenas uma vez, garantindo segurança e integridade.

ARQUITETURA



EXPLICAÇÃO DO CÓDIGO

CONTRACT

```
struct Proposal {  
    string name;  
    uint voteCount;  
}
```

- Define uma **estrutura (struct)** chamada Proposal com:
 - name: o nome da proposta (como o nome de um candidato).
 - voteCount: o número de votos recebidos.

```
Proposal[] public proposals;  
mapping(address => bool) public hasVoted;  
event Voted(address indexed voter, uint proposalIndex);
```


Cria um **array público** de propostas.

Cada elemento do array é do tipo Proposal.

Por ser public, o Solidity **automaticamente gera uma função getter**, permitindo que qualquer um leia os dados.

```
function getProposalsCount() public view returns (uint) {  ⛊ 2483 gas  
    return proposals.length;  
}
```

Função que retorna **quantas propostas existem**.


```
constructor(string[] memory _proposalNames) {  infinite gas 375400 gas  
    for (uint i = 0; i < _proposalNames.length; i++) {  
        proposals.push(Proposal({  
            name: _proposalNames[i],  
            voteCount: 0  
        }));  
    }  
}
```

Construtor: executado **uma vez** quando o contrato é implantado.

Recebe uma lista de nomes de propostas (como candidatos).

Para cada nome, cria uma nova Proposal com voteCount = 0 e a adiciona no array proposals.

```
function vote(uint _proposalIndex) public {  ⚙ infinite gas
    require(_proposalIndex < proposals.length, "Essa proposta nao existe.");
    require(!hasVoted[msg.sender], "Voce ja votou.");

    hasVoted[msg.sender] = true;
    proposals[_proposalIndex].voteCount++;
    emit Voted(msg.sender, _proposalIndex);
}
```

Função pública para permitir que qualquer pessoa vote.

Recebe o índice da proposta no array.

Verificações:

1. `require(_proposalIndex < proposals.length)`: impede que votem em propostas que não existem.
2. `require(!hasVoted[msg.sender])`: impede que o mesmo endereço vote mais de uma vez.

Ações:

- Marca `hasVoted[msg.sender] = true`.
- Incrementa `voteCount` da proposta escolhida.
- Emite o evento `Voted`.

FRONT-END

Como o front-end foi feito em react, então as partes estão separadas em diferentes arquivos



```
3
4 export const contractAddress = "0xDCd753568229B843Db5fddC151322b6884293F79";
5
6
7 export const contractABI = [
8   {
9     "inputs": [
10       {
11         "internalType": "string[]",
12         "name": "_proposalNames",
13         "type": "string[]"
14       }
15     ],
16     "stateMutability": "nonpayable",
17     "type": "constructor"
18   },
19   {
20     "inputs": [
21       {
22         "internalType": "uint256",
23         "name": "_proposalIndex",
24         "type": "uint256"
25       }
26     ],
```

Dentro do arquivo “contract-info.js” é passado o endereço e a ABI do contrato.

```
import { useState, useEffect, useCallback } from 'react';  
  
import { ethers } from 'ethers';  
import { contractAddress, contractABI } from './contracts/contract-info.js';  
import './App.css';
```

Aqui, no arquivo principal “APP.js”, está sendo importado as bibliotecas necessárias para a implementação do contrato.

```
const [account, setAccount] = useState(null);  
const [contract, setContract] = useState(null);  
const [proposals, setProposals] = useState([]);  
const [isLoading, setIsLoading] = useState(false);
```

- account: endereço da carteira conectada.
- contract: instância do contrato inteligente carregado com ethers.
- proposals: lista de propostas vindas do contrato.
- isLoading: indica se está carregando dados ou executando transação.

robots.txt
src
contracts
contract-info.js
App.css
App.js
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json
package.json
README.md

> OUTLINE

> TIMELINE

```
14 const loadProposals = useCallback(async () => {  
15   if (contract) {  
16     try {  
17       setIsLoading(true);  
18       const count = await contract.getProposalsCount();  
19       const tempProposals = [];  
20  
21       for (let i = 0; i < count; i++) {  
22         const proposal = await contract.proposals(i);  
23         tempProposals.push({  
24           name: proposal.name,  
25           voteCount: proposal.voteCount.toString()  
26         });  
27       }  
28  
29       setProposals(tempProposals);  
30     } catch (error) {  
31       console.error("Erro ao carregar propostas: ", error);  
32     } finally {  
33       setIsLoading(false);  
34     }  
35   }  
36 }, [contract]);
```

A função `loadProposals` busca as propostas do contrato inteligente e atualiza a interface com elas. Primeiro, ela verifica se o contrato foi carregado. Em seguida, ativa o estado de carregamento (`isLoading`), obtém a quantidade de propostas (`getProposalsCount`), e percorre cada uma delas acessando pelo índice com `contract.proposals(i)`.

Cada proposta é convertida para um objeto com `name` e `voteCount` (convertido para string) e armazenada em um array temporário. Ao final, esse array é usado para atualizar o estado `proposals`, o que faz o React exibir os dados na tela. Por fim, o carregamento é encerrado.


```
// useEffect para carregar as propostas quando o contrato estiver pronto
useEffect(() => {
  |   loadProposals();
}, [contract]);
```

Sempre que o contrato for carregado, essa função chama loadProposals().

```
const connectWallet = async () => {  
  if (window.ethereum) {  
    try {  
      const provider = new ethers.BrowserProvider(window.ethereum);  
      const signer = await provider.getSigner();  
      const address = await signer.getAddress();  
      const contractInstance = new ethers.Contract(contractAddress, contractABI, signer);  
  
      setAccount(address);  
      setContract(contractInstance);  
    } catch (err) {  
      console.error(err);  
    }  
  } else {  
    alert("Instale a MetaMask!");  
  }  
};
```

- Conecta à MetaMask.
- Usa ethers.BrowserProvider (nova forma no ethers v6).
- Obtém signer (quem vai enviar transações).
- Instancia o contrato com ethers.Contract.

```
// NOVA FUNÇÃO para lidar com o voto
const handleVote = async (proposalIndex) => {
  if (!contract) return;

  // Inicia o feedback visual para o usuário
  setIsLoading(true);
  try {
    // Chama a função 'vote' do nosso contrato inteligente
    const tx = await contract.vote(proposalIndex);
    // Espera a transação ser minerada e confirmada na blockchain
    await tx.wait();

    alert("Voto computado com sucesso!");
    // Recarrega os dados para mostrar o voto novo
    loadProposals();
  } catch (error) {
    console.error("Erro ao votar:", error);
    // Exibe a mensagem de erro que vem do 'require' do Solidity!
    alert(error.reason || "Ocorreu um erro ao processar seu voto.");
  } finally {
    // Termina o feedback visual
    setIsLoading(false);
  }
};
```

A função handleVote:

- Envia a transação de voto ao contrato.
- Espera a transação ser minerada.
- Em caso de erro, mostra error.reason (mensagem do require no Solidity).
- Após o voto, recarrega as propostas para mostrar os votos atualizados.

```
index.html
logo192.png
logo512.png
manifest.json
robots.txt
src
  contracts
    contract-info.js
  App.css
  App.js
  App.test.js
  index.css
  index.js
  logo.svg
  reportWebVitals.js
  setupTests.js
.gitignore
package-lock.json
package.json
README.md

OUTLINE
TIMELINE

89   return (
90     <div className="App">
91       <header className="App-header">
92         <h1>Sistema de Votação Descentralizado</h1>
93
94         {account ? (
95           <p>Conectado como: {account.substring(0, 6)}...{account.substring(account.length - 4)}</p>
96         ) : (
97           <button onClick={connectWallet} disabled={isLoading}>Conectar Carteira</button>
98         )}
99
100      <div className="proposals-section">
101        <h2>Candidatos:</h2>
102        {isLoading && <p>Processando transação, por favor aguarde...</p>}
103        {proposals.map((proposal, index) => (
104          <div key={index} className="proposal-card">
105            <span>{proposal.name}: {proposal.voteCount} votos</span>
106            { /* Adiciona o botão de voto e o desabilita durante o carregamento */ }
107            <button onClick={() => handleVote(index)} disabled={isLoading || !account}>
108              Votar
109            </button>
110          </div>
111        ))}
112      </div>
113    </header>
114  </div>
115  );
116 }
```

A interface do usuário mostra o título, o botão de conectar carteira ou o endereço. Assim como as propostas com botão de voto, e mostra uma mensagem de carregamento durante transações.