**Pasos Detallados para desarrollo de una Organización Autónoma Descentralizada (DAO) en una red de Blockchain Hyperledger**

Los siguientes pasos se han detallado para poder configurar y desplegar contratos inteligentes en una red Hyperledger Besu local en un MacBook Pro M1 (arm64).

1. **Instalación de Herramientas Necesarias**
   1. **Instalar Homebrew (si no lo tienes instalado)**

Homebrew te permite instalar y gestionar software de manera sencilla. Si aún no tienes Homebrew instalado en tu sistema, puedes hacerlo ejecutando el siguiente comando en la terminal.

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

* Este comando descargará y ejecutará un script de instalación de **Homebrew**. El script se obtiene de un repositorio oficial de GitHub (por eso usamos curl).
* El curl -fsSL es un comando que descarga el script desde la URL proporcionada.
* El script de instalación configurará **Homebrew** en tu sistema.

Después de ejecutar este comando, Homebrew se instalará en tu sistema. Podemos comprobar la correcta instalación con:

brew –versión

* 1. **Instalar Java 21**

[**https://www.oracle.com/java/technologies/downloads/#jdk21-mac**](https://www.oracle.com/java/technologies/downloads/#jdk21-mac)

java -version

Si Java 21 está instalado correctamente, deberías ver un mensaje similar que indica la versión de Java, algo como:

java version "21.0.5" 2024-10-15 LTS

Java(TM) SE Runtime Environment (build 21.0.5+9-LTS-239)  
Java HotSpot(TM) 64-Bit Server VM (build 21.0.5+9-LTS-239, mixed mode, sharing)

Puedes comprobar las versiones de Java instaladas en tu sistema (proporciona además ruta donde se ha instalado):

/usr/libexec/java\_home -V

Para configurar JAVA\_HOME, sigue estos pasos:

1. Abre el archivo de configuración de tu terminal (~/.zshrc en el caso de mac m1). Puedes hacerlo con el siguiente comando:

nano ~/.zshrc

1. Añade la variable JAVA\_HOME al final del archivo con esta línea:

export JAVA\_HOME=/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home

export PATH=$JAVA\_HOME/bin:$PATH

1. Guarda y cierra el archivo (en nano, presiona CTRL + O para guardar, luego CTRL + X para salir).
2. Recarga la configuración del archivo para aplicar los cambios:

source ~/.zshrc

1. Verifica la variable JAVA\_HOME:

echo $JAVA\_HOME

Debería mostrar la ruta de Java 21 que configuraste, confirmando que JAVA\_HOME está correctamente asignada.

* 1. **Hyperledger Besu**

[**https://besu.hyperledger.org/24.5.0/public-networks/get-started/install/run-docker-image**](https://besu.hyperledger.org/24.5.0/public-networks/get-started/install/run-docker-image)

1. **Desplegar Smart Contract en Hyperledger Besu usando Docker**
   1. **Instalar Docker**

Instalar Docker Desktop de la documentación oficial e instalarlo. Comprobar que funciona con:

docker –-version

* 1. **Configurar la red privada en Hyperledger Besu con Docker**
* Crear directorio para la red privada y acceder a el

mkdir besu-private-network

cd besu-private-network

* Crea un archivo genesis.json para definir la configuración de la red

nano genesis.json

cuyo contenido es:

{

"config":

{

"chainId": 2023,

"constantinopleBlock": 0,

"ethash": {}

},

"alloc":

{

"0xa9597c0ac6aa64ffae76087556b8e7e9a10d2073": {"balance": "1000000000000000000000"}

},

"difficulty": "0x1",

"gasLimit": "0x10000000"

}

* + - chainId=2023 para identificar nuestra red privada.
    - A9597c0AC6Aa64FfAE76087556B8E7E9a10D2073 es la cuenta de Metamask del chairperson
* Crea un archivo docker-compose.yml para desplegar Hyperledger Besu con Docker

nano docker-compose.yml

cuyo contenido es:

version: '3.8' # Especifica la versión de Docker Compose

services:

besu:

image: hyperledger/besu:latest # Imagen de Hyperledger Besu más reciente

container\_name: besu\_node # Nombre del contenedor

ports:

- 8545:8545 # Puerto RPC HTTP para interactuar con el nodo

# - 30303:30303 # Puerto P2P para la comunicación entre nodos, no es necesario, solo un nodo

# No define carpetas dentro del contenedor, sino qué carpetas están sincronizadas entre tu máquina y el contenedor

volumes:

- ./data:/var/lib/besu # En local (directorio donde hago docker-compose) tengo data y dentro del contenedor será accesible como /var/lib/besu

- ./genesis.json:/var/lib/besu/genesis.json # En local tengo genesis.json y el contenedor lo ve en /var/lib/besu/genesis.json

command:

- --network-id=2023 # Define el ID de la red Ethereum (único)

- --genesis-file=/var/lib/besu/genesis.json # Especifica el archivo de génesis (punto de arranque, configuración inicial del blockchain)

- --data-path=/var/lib/besu # Define la ruta donde se almacenan los datos del nodo

- --miner-enabled # Habilita la minería en el nodo

- --miner-coinbase=0xA9597c0AC6Aa64FfAE76087556B8E7E9a10D2073 # Dirección que recibirá las recompensas de minería (Cuenta Metamask)

- --min-gas-price=0 # Permite que las transacciones se procesen sin pagar gas.

- --rpc-http-enabled # Activa la API RPC HTTP para interactuar con el nodo (Metamask, frontend)

- --rpc-http-api=ETH,NET,WEB3,DEBUG,TXPOOL,ADMIN,MINER # Define qué APIs estarán habilitadas en RPC HTTP

- --host-allowlist="\*" # Permite accesos desde cualquier host (en producción, restringir)

- --rpc-http-cors-origins="all" # Permite peticiones desde cualquier origen (Es como decir, “acepta cualquier navegador”)

- --rpc-http-port=8545 # Expone la API HTTP en el puerto 8545.

- --p2p-enabled=false # Desactiva la comunicación P2P. Evita que el nodo busque pares.

- --sync-mode=FULL # Asegura la sincronización completa de la cadena.

* Levantar la red con Docker Compose

Antes de levantar la red asegúrate de tener iniciado Docker (arranca Docker Desktop) y posteriormente ejecuta:

docker-compose up -d

Esto descargará y ejecutará un nodo privado de Hyperledger Besu en segundo plano.

Comandos útiles Docker:

* **Listar todos los contenedores (incluidos los detenidos):**

docker ps -a

* **Ejecutar e iniciar un nuevo contenedor:**

docker run -d --name <container\_name> <image\_name>

-d: Ejecuta el contenedor en segundo plano (modo “detached”).

--name <container\_name>: Asigna un nombre al contenedor.

<image\_name>: Especifica la imagen a utilizar.

* **Detener un contenedor en ejecución:**

docker stop <container\_id>

* **Iniciar un contenedor que está detenido:**

docker start <container\_id>

* **Eliminar un contenedor específico:**

docker rm <container\_id>

* **Ver logs de un contenedor:**

docker logs <container\_id>

* **Acceder a la terminal de un contenedor en ejecución:**

docker exec -it <container\_id> /bin/bash

Si el contenedor tiene instalado Bash (que es lo más común en muchas imágenes base como Ubuntu, Debian, etc.)

* **Listar contenedores administrados por Docker Compose**

docker compose ps

* **Iniciar los servicios en segundo plano**

docker compose up -d

Nota: sin -d muestra los logs en cosnsola

* **Detener los contenedores sin eliminarlos:**

docker compose stop

* **Reiniciar servicios**

docker compose restart

* **Detener y eliminar los contenedores**

docker compose down

* **Ver los logs de los servicios**

docker compose logs -f

Nota: -f permite logs en tiempo real

1. **Metamask**
   1. **Conectar Metamask a la red**

* Abrir Metamask en Google Chrome (añadirlo como extensión).
* Crear una cuenta y guardar la contraseña y las 12 palabras claves
* Una vez dentro, abrir el desplegable de redes y pulsar en agregar una red personalizada que rellenaremos con los datos de nuestra red:
  + Nombre: El que queramos, en este caso ponemos “Besu local”
  + URL RPC: <http://localhost:8545/>
  + ID blockchain: 2023
  + Simbolo de moneda: ETH

Metamask se conectará a tu nodo besu (tiene que estar levantado el contenedor con docker-compose y activo)

1. **Instalar Yarn**

Instalar yarn como alternativa a npm,  con el objetivo de solucionar algunos problemas que tenía con npm en ese momento al instalar algunos paquetes.

brew install yarn

puedes comprobar su correcta instalación con:

yarn -v

1. **Usar Metamask y @truffle/hdwallet-provider**

Esto implica que el contrato se despliega desde Truffle usando una wallet (como MetaMask) a través de HDWalletProvider, y el frontend se conecta directamente a esa red.

* 1. **Crear un proyecto de truffle**

mkdir DAO-Dapp

cd DAO-Dapp

* Inicializar un nuevo proyecto Truffle

yarn init -y

Al ejecutarlo inicializar rápidamente un proyecto con Yarn como gestor de paquetes, creando un archivo package.json con valores predeterminados. Es equivalente a npm init pero usando yarn como gestor.

* Instalar dependencias

yarn add truffle web3 @truffle/hdwallet-provider

* Inicializar Truffle en el proyecto

npx truffle init

* 1. **Configurar Truffle para usar Hyperledger Besu**

Edita el archivo truffle-config.js y reemplázalo con:

// Permite firmar transacciones y conectarte al nodo Besu vía HTTP.

const HDWalletProvider = require("@truffle/hdwallet-provider");

const privateKeys = ["9fbae0d6b7a4fd39f0e8b0830f0e9952aa3bb98837f91ecc8b7b6164feec8a33"];

const providerURL = "http://127.0.0.1:8545";

module.exports = {

networks: {

besu: {

provider: () => new HDWalletProvider({

privateKeys: privateKeys,

providerOrUrl: providerURL,

numberOfAddresses: 1,

}),

network\_id: 2023,

gas: 7000000,

gasPrice: 0,

timeoutBlocks: 200,

networkCheckTimeout: 1000000,

},

},

// Versión de Solidity usada para compilar

compilers: {

solc: {

version: "0.8.0",

},

},

};

* 1. **Crear smart contract**

Dentro de my-smart-contract/contracts/, crea un archivo “nombreContrato.sol” con el contenido del contrato, en nuestro caso es “DAO.sol”.

Compilar el contrato con truffle:

npx truffle compile

Si todo está bien, verás una salida indicando que la compilación fue exitosa.

* 1. **Crear archivo de migración y despliegue del smart contract**

Para ello creamos en la carpeta migrations el archivo 2\_deploy\_contracts.js con el siguiente contenido:

// Importa el contrato DAO compilado para poder desplegarlo

const DAO = artifacts.require("DAO");

// Define la función que ejecutará Truffle para desplegar el contrato

module.exports = async function (deployer, network, accounts) {

// Define un array con la primera cuenta como miembro inicial del DAO

const initialMembers = [accounts[0]];

// Despliega el contrato DAO pasándole los miembros iniciales

await deployer.deploy(DAO, initialMembers);

};

Este script automatiza el despliegue del contrato DAO y asegura que tenga al menos un miembro inicial al momento de ser creado.

Posteriormente, realizamos la migración:

npx truffle migrate --network besu

Esto enviará la transacción y desplegará el contrato en tu red besu privada, especificada en el archivo truffle-config.js.

* truffle migrate: Ejecuta todas las migraciones disponibles en la carpeta migrations/ para desplegar contratos en la blockchain.
* --network besu: Especifica la red donde se realizará la migración. Debe estar definida en truffle-config.js.

Si todo va bien verás una salida como la siguiente:

Deploying 'DAO'

DAO: 0xabc123... (dirección del contrato)

1. **Frontend con React + Vite + Metamask**
   1. **Crear frontend con Vite + React**

Desde la raíz del proyecto Truffle (DAO-DApp), ejecutamos:

yarn create vite frontend --template react

siendo:

* **yarn create:** Es un atajo para ejecutar generadores de proyectos.
* **vite:** Especifica qué generador se quiere usar. En este caso: create-vite, que genera proyectos nuevos usando Vite. Vite es un bundler ultrarrápido para desarrollo frontend moderno.
* **frontend:** Es el nombre del directorio que se va a crear.Todos los archivos del proyecto se guardarán en una carpeta llamada frontend.
* **--template react:** Indica que se va a usar como plantilla predefinida React.

Luego:

cd frontend

yarn

El comando “yarn” instala todas las dependencias listadas en frontend/package.json, es decir:

• React

• Vite

• Dependencias internas del proyecto

Nota: El comando yarn dev inicia el servidor de desarrollo, compila el código y levanta una web local para poder ver y trabajar en la app en tiempo real.

Posteriormente, instalamos Web3 con:

yarn add web3

Ahora, procedemos a la implementación de los distintos componentes que van a otorgar funcionalidad a nuestro frontend y que van a permitirnos interactuar a través de Google Chrome con el Smart contract desplegado.

Esta lógica estará implementada dentro de la carpeta src dentro de nuestro frontend. A continuación, se detallan los componentes más importantes:

1. En primer lugar, implementamos src/getWeb3.js cuyo objetivo es el de inicializar una instancia de Web3 para conectar con MetaMask o, si no está disponible, usar un nodo local (como el de Hyperledger Besu en localhost:8545).

// src/getWeb3.js

import Web3 from "web3";

async function getWeb3() {

// Comprobar si Metamask u otra wallet compatible está instalada y activa.

if (window.ethereum) {

const web3 = new Web3(window.ethereum);

try {

await window.ethereum.request({ method: "eth\_requestAccounts" });

} catch (error) {

console.error("User denied account access or error occurred:", error);

throw error;

}

return web3;

} else {

// Navegador no tiene Metamask

console.warn("No Metamask detected, using fallback to local provider.");

// Aunque no se pueda firmar transacciones (no Metamask), se podrá leer datos o interactuar en modo lectura

return new Web3("http://127.0.0.1:8545");

}

}

export default getWeb3;

* Verifica si window.ethereum está disponible (MetaMask).
* Solicita acceso a las cuentas (eth\_requestAccounts).
* Si no hay MetaMask, crea un Web3 apuntando a http://127.0.0.1:8545.
* Se exporta la instancia para usarla en toda la app.

1. En segundo lugar, crear src/App.jsx. Es el componente principal de la aplicación, orquesta toda la lógica de conexión e interacción con el smart contract. Se encarga de:
   * Inicializar Web3.
   * Detectar cuentas y red.
   * Cargar el contrato DAO desde Truffle.
   * Configurar la navegación entre vistas con React Router.

import React, { useEffect, useState, useRef } from "react";

import { BrowserRouter as Router, Routes, Route, Link, Navigate } from "react-router-dom";

import getWeb3 from "./getWeb3";

import DAOABI from "../../build/contracts/DAO.json";

// Rutas

import Home from "./routes/Home";

import Proposals from "./routes/Proposals";

import CreateProposal from "./routes/CreateProposal";

import AddMember from "./routes/AddMember";

import Members from "./routes/Members";

// Estilos globales

import "./App.css";

// Importa icono

import DAOIcon from "./assets/icon.png";

function App() {

const [web3, setWeb3] = useState(null);

const [contract, setContract] = useState(null);

const [accounts, setAccounts] = useState([]);

const [chairperson, setChairperson] = useState("");

const [memberCount, setMemberCount] = useState(0);

const [isMember, setIsMember] = useState(false);

const [isLoading, setIsLoading] = useState(true);

const cleanupRef = useRef(null);

async function updateMemberStatus(contractInstance, account) {

if (contractInstance && account) {

try {

const memberStatus = await contractInstance.methods.isMember(account).call();

setIsMember(memberStatus);

} catch (error) {

console.error("Error verificando el estado de miembro:", error);

setIsMember(false);

}

}

}

useEffect(() => {

let instance;

async function init() {

try {

const web3Instance = await getWeb3();

setWeb3(web3Instance);

const accountsList = await web3Instance.eth.getAccounts();

setAccounts(accountsList);

const networkId = await web3Instance.eth.net.getId();

const deployedNetwork = DAOABI.networks[networkId];

if (!deployedNetwork) {

alert("Contrato no desplegado en esta red");

setIsLoading(false);

return;

}

instance = new web3Instance.eth.Contract(DAOABI.abi, deployedNetwork.address);

setContract(instance);

const chair = await instance.methods.chairperson().call();

setChairperson(chair);

const memCount = await instance.methods.countMembers().call();

setMemberCount(memCount);

await updateMemberStatus(instance, accountsList[0]);

if (window.ethereum) {

const handleAccountsChanged = async (accountsChanged) => {

if (accountsChanged.length > 0) {

setAccounts(accountsChanged);

if (instance) {

try {

const chair = await instance.methods.chairperson().call();

setChairperson(chair);

const memCount = await instance.methods.countMembers().call();

setMemberCount(memCount);

await updateMemberStatus(instance, accountsChanged[0]);

} catch (error) {

console.error("Error actualizando estado tras cambio de cuenta:", error);

}

}

} else {

setAccounts([]);

setIsMember(false);

}

};

const handleChainChanged = () => {

window.location.reload();

};

window.ethereum.on("accountsChanged", handleAccountsChanged);

window.ethereum.on("chainChanged", handleChainChanged);

cleanupRef.current = () => {

window.ethereum.removeListener("accountsChanged", handleAccountsChanged);

window.ethereum.removeListener("chainChanged", handleChainChanged);

};

}

} catch (error) {

console.error("Error al inicializar:", error);

} finally {

setIsLoading(false);

}

}

init();

return () => {

if (cleanupRef.current) {

cleanupRef.current();

}

};

}, []);

useEffect(() => {

if (contract && accounts.length > 0) {

updateMemberStatus(contract, accounts[0]);

}

}, [accounts, contract]);

if (isLoading) return (

<div className="container">

<div className="card">

<p style={{ fontSize: "1.5rem" }}>Cargando DAO...</p>

</div>

</div>

);

return (

<Router>

<nav>

<Link to="/">

<img

src={DAOIcon}

alt="DAO icon"

style={{ width: "40px", height: "40px", marginRight: "0.75rem", verticalAlign: "middle" }}

/>

DAO - TFM 2025

</Link>

{accounts[0] === chairperson && <Link to="/add-member">Agregar Miembro</Link>}

<Link to="/members">Miembros</Link>

{isMember && <Link to="/create-proposal">Crear Propuesta</Link>}

<Link to="/proposals">Propuestas</Link>

</nav>

<Routes>

<Route path="/" element={

<Home

accounts={accounts}

chairperson={chairperson}

memberCount={memberCount}

isMember={isMember}

/>

} />

<Route path="/proposals" element={

<Proposals

contract={contract}

account={accounts[0]}

isMember={isMember}

chairperson={chairperson}

web3={web3}

/>

} />

<Route path="/create-proposal" element={

isMember ? (

<CreateProposal contract={contract} account={accounts[0]} web3={web3} />

) : (

<Navigate to="/" replace />

)

} />

<Route path="/members" element={

<Members contract={contract} account={accounts[0]} />

} />

<Route path="/add-member" element={

accounts[0] === chairperson ? (

<AddMember contract={contract} account={accounts[0]} web3={web3} />

) : (

<Navigate to="/" replace />

)

} />

<Route path="\*" element={

<div className="container">

<div className="card">

<p>404: Página no encontrada</p>

</div>

</div>

} />

</Routes>

</Router>

);

}

export default App;

* Usa useState para manejar el estado de la cuenta, contrato, etc.
* Usa useEffect para iniciar Web3 y configurar eventos de MetaMask (accountsChanged, chainChanged).
* Carga el contrato DAO desde DAO.json, según el networkId.
* Protege rutas según permisos (solo miembros o chairperson pueden ver ciertas vistas).
* Implementa un sistema de rutas: Home, Proposals, CreateProposal, AddMember, Members, etc.

1. Posteriormente, hay que implementar las diferentes “routes” que se va a encargar de manejar el “router” descrito en App.jx. Para ello, creamos la carpeta src/routes y en ella almacenaremos los distintos componentes.
2. Nuestra primera “route” será Home.jsx, que será el punto de partida en nuestro frontend, se encargará de mostrar información general sobre la DAO al usuario conectado.

import React from "react";

import Layout from "../components/Layout";

// Imports iconos

import Information from "../assets/information.png";

function Home({ accounts, chairperson, memberCount, isMember }) {

const currentAccount = accounts && accounts.length > 0 ? accounts[0] : "No conectada";

return (

<Layout>

<h2 style={{ display: "flex", alignItems: "center", gap: "0.5rem" }}>

<img

src={Information}

alt="Home"

style={{ width: "40px", height: "40px", verticalAlign: "middle" }}

/>

Información de la DAO

</h2>

<div className="info-block">

<p><span className="label">Cuenta conectada:</span><br />{currentAccount}</p>

<p><span className="label">Chairperson:</span><br />{chairperson || "Cargando..."}</p>

<p><span className="label">Número de miembros:</span><br />{memberCount}</p>

<p><span className="label">Eres miembro:</span><br />{isMember ? "Sí" : "No"}</p>

{!isMember && (

<p className="warning">No eres miembro de esta DAO.</p>

)}

</div>

</Layout>

);

}

export default Home;

* Muestra la cuenta conectada.
* Muestra la dirección del chairperson.
* Muestra número total de miembros.
* Muestra si el usuario es miembro o no.
* Usa el componente Layout como envoltorio común.

1. Continuamos con las diferentes vistas que completan nuestro frontend y que recogen la funcionalidad implementada en el Smart contract DAO.sol

Proposals.jsx: Lista todas las propuestas creadas en la DAO y permitir votar a favor o en contra.

import React, { useEffect, useState } from "react";

import Layout from "../components/Layout";

// Icons

import Execute from "../assets/execute.png";

import Pending from "../assets/pending.png";

import Results from "../assets/results.png";

function Proposals({ contract, account, isMember, chairperson, web3 }) {

const [proposals, setProposals] = useState([]);

const [isMining, setIsMining] = useState(false);

useEffect(() => {

if (!contract || !account) return;

loadProposals();

}, [contract, account]);

async function loadProposals() {

try {

const count = await contract.methods.getProposalsCount().call();

const loaded = [];

for (let i = 0; i < count; i++) {

try {

const proposal = await contract.methods.getProposal(i).call({ from: account });

loaded.push({

index: i,

description: proposal.description,

options: proposal.options,

voteCounts: proposal.voteCounts.map(Number),

executed: proposal.executed,

hasVoted: proposal.hasVoted,

winningOption:

proposal.executed && Number(proposal.winningVotes) > 0

? { option: proposal.winningOption, votes: Number(proposal.winningVotes) }

: null,

});

} catch (innerErr) {

console.error(`❌ Error al obtener propuesta #${i}:`, innerErr);

}

}

setProposals(loaded);

} catch (err) {

console.error("❌ Error general al cargar propuestas:", err);

}

}

async function vote(proposalIndex, optionIndex) {

try {

setIsMining(true);

const gasPrice = await web3.eth.getGasPrice();

await contract.methods.vote(proposalIndex, optionIndex).send({

from: account,

gas: 200000,

gasPrice,

});

await loadProposals();

} catch (err) {

console.error("❌ Error al votar:", err);

const reason = err?.message?.match(/reason=\"([^\"]+)\"/)?.[1] || err.message;

alert(`❌ Error al votar: ${reason}`);

} finally {

setIsMining(false);

}

}

async function execute(index) {

try {

setIsMining(true);

const gasPrice = await web3.eth.getGasPrice();

await contract.methods.executeProposal(index).send({

from: account,

gas: 200000,

gasPrice,

});

await loadProposals();

} catch (err) {

console.error("❌ Error al ejecutar propuesta:", err);

const reason = err?.message?.match(/reason=\"([^\"]+)\"/)?.[1] || err.message;

alert(`❌ No se pudo ejecutar: ${reason}`);

} finally {

setIsMining(false);

}

}

const pendingProposals = proposals.filter((p) => !p.executed);

const executedProposals = proposals.filter((p) => p.executed);

return (

<Layout>

{isMining && <p>⛏️ Procesando transacción en la blockchain...</p>}

<div style={{ display: "flex", gap: "6rem", alignItems: "flex-start", flexWrap: "wrap" }}>

{/\* Propuestas pendientes \*/}

<div style={{ flex: 1, minWidth: "400px" }}>

<h2 style={{ display: "flex", alignItems: "center", gap: "0.5rem", marginBottom: "1rem" }}>

<img src={Pending} alt="Pendientes" style={{ width: "40px", height: "40px" }} />

Propuestas pendientes

</h2>

{pendingProposals.length === 0 ? (

<p>No hay propuestas pendientes.</p>

) : (

<ul style={{ listStyle: "none", padding: 0 }}>

{pendingProposals.map((p) => {

const totalVotes = p.voteCounts.reduce((a, b) => a + b, 0);

return (

<li

key={p.index}

style={{

marginBottom: "1rem",

padding: "1.7rem",

border: "1px solid #ccc",

borderRadius: "10px",

backgroundColor: "#f5f5f5",

}}

>

<strong style={{ fontSize: "1.2rem" }}>

{p.index}: {p.description}

</strong>

<ul style={{ paddingLeft: 0, marginTop: "1rem" }}>

{p.options.map((opt, idx) => {

const count = p.voteCounts[idx];

const percentage = totalVotes > 0 ? (count / totalVotes) \* 100 : 0;

return (

<li key={idx} style={{ marginBottom: "1.5rem" }}>

<div style={{ marginBottom: "0.3rem" }}>{opt}</div>

<div

style={{

backgroundColor: "#ddd",

borderRadius: "6px",

overflow: "hidden",

height: "20px",

}}

>

<div

style={{

width: `${percentage}%`,

backgroundColor: "#007bff",

height: "100%",

}}

/>

</div>

<div

style={{

display: "flex",

justifyContent: "space-between",

alignItems: "center",

marginTop: "0.5rem",

}}

>

<span style={{ color: "#555" }}>{count} votos</span>

{!p.hasVoted && isMember && (

<button

onClick={() => vote(p.index, idx)}

disabled={isMining}

style={{

backgroundColor: "#007bff",

color: "white",

border: "none",

borderRadius: "6px",

padding: "0.4rem 0.8rem",

cursor: isMining ? "not-allowed" : "pointer",

}}

>

Votar

</button>

)}

</div>

</li>

);

})}

</ul>

{p.hasVoted && (

<p style={{ color: "gray", fontStyle: "italic" }}>

Ya votaste en esta propuesta

</p>

)}

{account === chairperson && (

<button

onClick={() => execute(p.index)}

disabled={isMining}

style={{

marginTop: "10px",

backgroundColor: "#007bff",

color: "white",

border: "none",

borderRadius: "6px",

padding: "0.5rem 1rem",

cursor: isMining ? "not-allowed" : "pointer",

}}

>

Ejecutar propuesta

</button>

)}

</li>

);

})}

</ul>

)}

</div>

{/\* Propuestas ejecutadas \*/}

<div style={{ flex: 1, minWidth: "400px" }}>

<h2 style={{ display: "flex", alignItems: "center", gap: "0.5rem", marginBottom: "1rem" }}>

<img src={Execute} alt="Ejecutadas" style={{ width: "40px", height: "40px" }} />

Propuestas ejecutadas

</h2>

{executedProposals.length === 0 ? (

<p>No hay propuestas ejecutadas aún.</p>

) : (

<ul style={{ listStyle: "none", padding: 0 }}>

{executedProposals.map((p) => (

<li

key={p.index}

style={{

marginBottom: "1rem",

padding: "1.7rem",

border: "1px solid #ccc",

borderRadius: "10px",

backgroundColor: "#eaeaea",

}}

>

<strong style={{ fontSize: "1.2rem" }}>

{p.index}: {p.description}

</strong>

{p.winningOption ? (

<p style={{ marginTop: "0.5rem", color: "black", display: "flex", alignItems: "center", gap: "0.5rem" }}>

<img src={Results} alt="Resultado" style={{ width: "25px", height: "25px" }} />

Resultado: <strong>{p.winningOption.option}</strong> (

{p.winningOption.votes} votos)

</p>

) : (

<p style={{ marginTop: "0.5rem", color: "gray" }}>

La propuesta fue ejecutada, pero no recibió votos.

</p>

)}

</li>

))}

</ul>

)}

</div>

</div>

</Layout>

);

}

export default Proposals;

* Llama al método getProposal(i) para cada propuesta.
* Usa isMember para mostrar botones de votación.
* Verifica si la propuesta aún está abierta.
* Usa contract.methods.vote(...) para emitir votos.

CreateProposal.jsx: Permitir a un miembro crear una nueva propuesta, indicando una descripción acerca de esta y con las diferentes opciones a votar.

import React, { useState } from "react";

import { useNavigate } from "react-router-dom";

import Layout from "../components/Layout";

// Imports iconos

import CreateProposalIcon from "../assets/createproposal.png";

import X from "../assets/x.png";

function CreateProposal({ contract, account, web3 }) {

const [proposalDescription, setProposalDescription] = useState("");

const [options, setOptions] = useState(["", ""]);

const [isMining, setIsMining] = useState(false);

const navigate = useNavigate();

const handleOptionChange = (index, value) => {

const newOptions = [...options];

newOptions[index] = value;

setOptions(newOptions);

};

const addOptionField = () => {

setOptions([...options, ""]);

};

const removeOptionField = (index) => {

const updated = options.filter((\_, i) => i !== index);

setOptions(updated);

};

const createProposal = async () => {

const cleanOptions = options.map(opt => opt.trim()).filter(opt => opt !== "");

const hasDuplicates = new Set(cleanOptions).size !== cleanOptions.length;

if (!proposalDescription.trim() || cleanOptions.length < 2) {

alert("Debes ingresar una descripción y al menos dos opciones.");

return;

}

if (hasDuplicates) {

alert("Las opciones deben ser únicas.");

return;

}

try {

setIsMining(true);

const gasPrice = await web3.eth.getGasPrice();

const tx = await contract.methods

.createProposal(proposalDescription, cleanOptions)

.send({

from: account,

gas: 300000,

gasPrice,

});

console.log("✅ Transacción de creación completada:", tx);

alert("✅ Propuesta creada con éxito.");

// Redirige a /proposals tras éxito

navigate("/proposals");

} catch (error) {

console.error("❌ Error al crear propuesta:", error);

const reason = error?.message?.match(/reason="([^"]+)"/)?.[1] || error.message;

alert(`❌ Error al crear propuesta: ${reason}`);

} finally {

setIsMining(false);

}

};

return (

<Layout>

<h2 style={{ display: "flex", alignItems: "center", gap: "0.5rem" }}>

<img

src={CreateProposalIcon}

alt="Crear propuesta"

style={{ width: "40px", height: "40px", verticalAlign: "middle" }}

/>

Crear nueva propuesta

</h2>

<input

type="text"

value={proposalDescription}

onChange={(e) => setProposalDescription(e.target.value)}

placeholder="Descripción de la propuesta"

disabled={isMining}

style={{

width: "100%",

padding: "0.6rem",

fontSize: "1rem",

borderRadius: "6px",

border: "1px solid #ccc",

backgroundColor: "#f5f5f5",

color: "#333",

marginBottom: "1rem"

}}

/>

<h4 style={{ marginBottom: "0.5rem" }}>Opciones:</h4>

{options.map((opt, idx) => (

<div key={idx} style={{ display: "flex", gap: "10px", marginBottom: "0.5rem" }}>

<input

type="text"

value={opt}

onChange={(e) => handleOptionChange(idx, e.target.value)}

placeholder={`Opción ${idx + 1}`}

disabled={isMining}

style={{

flex: 1,

padding: "0.5rem",

fontSize: "1rem",

borderRadius: "6px",

border: "1px solid #ccc",

backgroundColor: "#f5f5f5",

color: "#333"

}}

/>

{options.length > 2 && (

<button

type="button"

onClick={() => removeOptionField(idx)}

disabled={isMining}

style={{

background: "transparent",

border: "none",

padding: "0.3rem",

cursor: isMining ? "not-allowed" : "pointer",

}}

title="Quitar opción"

>

<img

src={X}

alt="Eliminar"

style={{

width: "25px",

height: "25px",

display: "block",

}}

/>

</button>

)}

</div>

))}

<button

onClick={addOptionField}

disabled={isMining}

style={{

marginTop: "1rem",

padding: "0.5rem 1rem",

borderRadius: "6px",

fontSize: "1rem",

backgroundColor: "#007bff",

color: "white",

border: "none",

cursor: isMining ? "not-allowed" : "pointer"

}}

>

Añadir opción

</button>

<br />

<button

onClick={createProposal}

disabled={isMining}

style={{

marginTop: "1rem",

padding: "0.6rem 1.2rem",

borderRadius: "6px",

fontSize: "1rem",

backgroundColor: "#007bff",

color: "white",

border: "none",

cursor: isMining ? "not-allowed" : "pointer"

}}

>

Crear Propuesta

</button>

{isMining && <p style={{ marginTop: "1rem" }}>⛏️ Minando transacción...</p>}

</Layout>

);

}

export default CreateProposal;

* Formulario controlado con useState.
* Valida que se complete el formulario.
* Convierte el valor a wei (web3.utils.toWei) antes de enviarlo.
* Usa contract.methods.createProposal(...) para enviarlo.

AddMember.jsx: Permitir al chairperson agregar nuevas direcciones como miembros de la DAO.

import React, { useState } from "react";

import { useNavigate } from "react-router-dom";

import Layout from "../components/Layout";

// Import icono

import AddUser from "../assets/adduser.png";

function AddMember({ contract, account, web3 }) {

const [newMemberAddress, setNewMemberAddress] = useState("");

const [isMining, setIsMining] = useState(false);

const [statusMessage, setStatusMessage] = useState("");

const navigate = useNavigate();

const handleChange = (e) => {

setNewMemberAddress(e.target.value);

setStatusMessage(""); // Limpiar mensaje anterior

};

const addMember = async () => {

if (!web3.utils.isAddress(newMemberAddress)) {

setStatusMessage("❌ Dirección Ethereum inválida.");

return;

}

try {

setIsMining(true);

setStatusMessage("⛏️ Enviando transacción...");

await contract.methods.addMember(newMemberAddress).send({

from: account,

gas: 100000,

gasPrice: "0",

});

setStatusMessage("✅ Miembro agregado con éxito.");

setNewMemberAddress("");

// Redirige a /members tras éxito

navigate("/members");

} catch (error) {

console.error("Error al agregar miembro:", error);

setStatusMessage("❌ Error al agregar miembro.");

} finally {

setIsMining(false);

}

};

return (

<Layout>

<h2 style={{ display: "flex", alignItems: "center", gap: "0.5rem" }}>

<img

src={AddUser}

alt="Agregar usuario"

style={{ width: "40px", height: "40px", verticalAlign: "middle" }}

/>

Agregar nuevo miembro

</h2>

<input

type="text"

value={newMemberAddress}

onChange={handleChange}

placeholder="Introduce la dirección del nuevo miembro"

style={{

width: "100%",

padding: "0.6rem",

fontSize: "1rem",

borderRadius: "6px",

border: "1px solid #ccc",

marginBottom: "1rem",

backgroundColor: "#f5f5f5",

color: "#333",

}}

/>

<br />

<button

onClick={addMember}

disabled={isMining}

style={{

padding: "0.6rem 1.2rem",

fontSize: "1rem",

borderRadius: "6px",

border: "none",

backgroundColor: isMining ? "#999" : "#007bff",

color: "white",

cursor: isMining ? "not-allowed" : "pointer",

transition: "background-color 0.2s",

}}

>

Agregar Miembro

</button>

{statusMessage && (

<p style={{ marginTop: "1rem", fontWeight: "bold" }}>{statusMessage}</p>

)}

</Layout>

);

}

export default AddMember;

* Solo el chairperson puede acceder a esta ruta (controlado desde App.jsx).
* Permite ingresar una dirección Ethereum válida.
* Llama a contract.methods.addMember(...).

Members.jsx: Listar todas las direcciones de los miembros actuales de la DAO.

import React, { useState, useEffect } from "react";

import Web3 from "web3";

import Layout from "../components/Layout";

// Imports iconos

import MembersIcon from "../assets/members.png";

import Admin from "../assets/admin.png";

import DeleteUser from "../assets/deleteuser.png";

function Members({ contract, account }) {

const [members, setMembers] = useState([]);

const [status, setStatus] = useState("");

const [isMining, setIsMining] = useState(false);

const [chairperson, setChairperson] = useState("");

useEffect(() => {

async function fetchData() {

try {

const count = await contract.methods.countMembers().call();

const loaded = [];

for (let i = 0; i < count; i++) {

const member = await contract.methods.memberList(i).call();

loaded.push(member);

}

const chair = await contract.methods.chairperson().call();

setChairperson(chair);

setMembers(loaded);

} catch (error) {

console.error("Error cargando miembros:", error);

}

}

if (contract) fetchData();

}, [contract]);

const handleRemove = async (address) => {

try {

if (!window.ethereum) {

setStatus("❌ No se encontró provider (Metamask).");

return;

}

// Instancia local solo para utilidades y llamadas web3

const web3 = new Web3(window.ethereum);

// Validar con la librería

if (!Web3.utils.isAddress(address)) {

setStatus("❌ Dirección inválida.");

return;

}

// Asegura permisos de cuenta

await window.ethereum.request({ method: "eth\_requestAccounts" });

setIsMining(true);

setStatus(`⛏️ Eliminando miembro ${address}...`);

// Prepara la transacción

const tx = contract.methods.removeMember(address);

// Estimar gas con los mismos parámetros que usaremos

const gas = await tx.estimateGas({ from: account });

// Forzar tx tipo 0 (legacy) en redes sin EIP-1559

const gasPrice = await web3.eth.getGasPrice();

await tx.send({

from: account,

gas,

gasPrice,

});

setStatus(`✅ Miembro ${address} eliminado.`);

setMembers((prev) =>

prev.filter((addr) => addr.toLowerCase() !== address.toLowerCase())

);

} catch (error) {

console.error(error);

setStatus(`❌ Error al eliminar miembro. ${error?.message ?? ""}`);

} finally {

setIsMining(false);

}

};

const isChairperson = account?.toLowerCase() === chairperson?.toLowerCase();

return (

<Layout>

<h2 style={{ display: "flex", alignItems: "center", gap: "0.5rem" }}>

<img

src={MembersIcon}

alt="Miembros"

style={{ width: "40px", height: "40px", verticalAlign: "middle" }}

/>

Miembros actuales

</h2>

{members.length === 0 ? (

<p>No hay miembros registrados.</p>

) : (

<ul style={{ listStyle: "none", paddingLeft: 0 }}>

{members.map((addr, idx) => (

<li

key={idx}

style={{

display: "flex",

justifyContent: "space-between",

alignItems: "center",

padding: "0.75rem 0",

borderBottom: "1px solid #eee",

}}

>

{/\* Dirección alineada a la izquierda con espacio a la derecha \*/}

<div

style={{

flex: 1,

wordBreak: "break-all",

marginRight: "2rem", // Espacio entre dirección e iconos

}}

>

{addr}

</div>

{/\* Iconos alineados a la derecha \*/}

<div style={{ display: "flex", alignItems: "center", gap: "1rem" }}>

{addr.toLowerCase() === chairperson?.toLowerCase() && (

<img

src={Admin}

alt="Admin"

title="Chairperson"

style={{ width: "30px", height: "30px" }}

/>

)}

{isChairperson && addr.toLowerCase() !== chairperson?.toLowerCase() && (

<button

style={{

background: "transparent",

border: "none",

cursor: isMining ? "not-allowed" : "pointer",

padding: 0,

}}

onClick={() => handleRemove(addr)}

disabled={isMining}

title="Eliminar miembro"

>

<img

src={DeleteUser}

alt="Eliminar"

style={{ width: "30px", height: "30px" }}

/>

</button>

)}

</div>

</li>

))}

</ul>

)}

{status && (

<p style={{ marginTop: "1rem", fontWeight: "bold" }}>{status}</p>

)}

</Layout>

);

}

export default Members;

* Llama a countMembers() para saber cuántos hay.
* Itera de 0 a N y obtiene members(i).
* Muestra cada dirección en pantalla.

Nota: Para poder aplicar estilos, se ha creado un componente Layout.jsx dentro de src/components para diseñar un envoltorio común para todas las vistas, con padding y estilo uniforme.

// src/components/Layout.jsx

import React from "react";

function Layout({ children }) {

return (

<div className="container">

<div className="card">

{children}

</div>

</div>

);

}

export default Layout;

Estos estilos se definen en App.css dentro de src.