

Sistema de Votación para una Comunidad usando OpenZeppelin Sprint 4: Proyecto FINAL

Descripción

Este proyecto es un sistema de votación descentralizado para una comunidad, implementado utilizando un smart contract en Ethereum y una interfaz de usuario desarrollada en React. Permite a los miembros registrados votar en propuestas y ver los resultados de las votaciones de manera transparente y segura.

Tecnologías Utilizadas

- **Solidity:** Lenguaje de programación para escribir smart contracts.
- **Ethereum Testnet (Sepolia):** Red de prueba donde se despliega el contrato.
- **MetaMask:** Wallet de Ethereum para interactuar con el blockchain.
- **Remix:** IDE de Ethereum usado para desarrollar y desplegar el smart contract.
- **Visual Studio Code:** Editor de código utilizado para el desarrollo del frontend.
- **React:** Biblioteca de JavaScript para construir la interfaz de usuario.
- **Web3.js:** Biblioteca para interactuar con nodos Ethereum desde el navegador.
- **Material-UI:** Biblioteca de componentes React para un diseño de interfaz moderno y responsive.

Mejora del proyecto final

1. Control de errores del smart contract

La mejora realizada en la versión modificada del contrato de Solidity Community es la inclusión de una verificación adicional en varias funciones para asegurarse de que solo el propietario del contrato puede ejecutarlas. Esta verificación se realiza mediante la condición **`require(msg.sender == owner(), "Error: Solo el propietario puede ejecutar esta funcion")`**; añadida en las funciones `registerMember`, `createProposal`, y `executeProposal`. Esta es una medida de seguridad adicional que garantiza que las operaciones críticas no puedan ser ejecutadas por una cuenta no autorizada.

La versión original de estas funciones utiliza solo el modificador `onlyOwner` de OpenZeppelin, que ya impide que usuarios que no son el propietario ejecuten estas funciones. Sin embargo, la inclusión explícita de la verificación `require` con un mensaje de error personalizado puede proporcionar una retroalimentación más clara a quien intenta ejecutar la función sin ser el propietario, mejorando la experiencia del usuario y haciendo el código más legible en cuanto a sus requerimientos de seguridad.

Además, estas adiciones hacen que las intenciones de seguridad y control del contrato sean explícitas, lo que es una práctica recomendada en el desarrollo de contratos inteligentes para evitar errores o malentendidos sobre los controles de acceso que están en vigencia.

2. Mejoras al código react

En la nueva versión del código React que proporcionaste, se ha añadido una funcionalidad clave que extiende la interactividad del sistema de votación de comunidades. Aquí detallo las funciones adicionales y cómo enriquecen la aplicación:

Funciones Adicionales Incorporadas

Ejecutar Propuesta (Execute Proposal):

- Descripción: Se ha implementado una nueva función `executeProposal`, que permite ejecutar una propuesta ya existente basada en su ID. Esta función es crucial para activar el proceso de votación o cualquier otra acción que deba realizarse una vez que una propuesta haya sido aprobada o necesite activarse.
- Impacto: Permite al administrador o a los usuarios autorizados a llevar a cabo acciones decisivas sobre las propuestas, facilitando una gestión efectiva y dinámica dentro de la plataforma de votación.

Menú Ampliado:

- Descripción: Se ha añadido una nueva opción en el menú, "Opcion 3: Ejecutar o activar Encuesta por numero", que se vincula directamente con la función `executeProposal`.
- Impacto: Proporciona a los usuarios un acceso rápido y fácil para ejecutar propuestas directamente desde el menú principal, mejorando la usabilidad y accesibilidad de la aplicación.

Cómo Funcionan Estas Adiciones

Interactividad con el Menú:

- Cuando se selecciona la opción "Ejecutar o activar Encuesta por numero" del menú, la función `handleMenuItemClick` detecta esta elección y ejecuta la función `executeProposal`.
- Esto desencadena una llamada al contrato inteligente en la blockchain para llevar a cabo la acción designada para la propuesta especificada.

Integración con Blockchain:

- Utilizando `Web3.js`, esta funcionalidad interactúa con el contrato inteligente desplegado en la blockchain de Ethereum, enviando una transacción que puede incluir cambios en el estado del contrato basado en la lógica definida para la ejecución de la propuesta.

Implicaciones Técnicas y de Usuario

- Experiencia de Usuario Mejorada: Los usuarios ahora pueden gestionar propuestas de manera más efectiva, desde su creación hasta su ejecución, todo dentro de la misma interfaz.
- Automatización de Procesos: Al permitir la ejecución automática de propuestas, se reduce la necesidad de intervención manual, lo que puede minimizar errores y aumentar la eficiencia de los procesos de toma de decisiones.
- Seguridad y Control: La ejecución de propuestas a través de transacciones blockchain asegura que todas las acciones son registradas, verificables y seguras, lo que es vital para mantener la integridad de un sistema de votación.

Estas adiciones hacen que la aplicación no solo sea un lugar para votar sobre las propuestas, sino también un centro de gestión de propuestas donde se pueden llevar a cabo acciones críticas basadas en los resultados de las votaciones o en las necesidades de la comunidad. Esta integración más profunda con el

contrato inteligente subyacente expande las capacidades del sistema para facilitar una administración eficaz y transparente de actividades comunitarias.

3. Se compila, se despliega y valida en Sepolia Test

- **URL Contrato Validado**

<https://sepolia.etherscan.io/address/0x283bbb7fc922f0d82501e5c4292f4dd3ba068d8d#code>

- **Salida de Truffle usando Truffle**

```
ubuntu@ubuntu:~/Documents/GitHub/mi-dao-dapp$ truffle migrate --network dashboard
```

```
Compiling your contracts...
```

```
=====
```

```
> Everything is up to date, there is nothing to compile.
```

```
Starting migrations...
```

```
=====
```

```
> Network name:      'dashboard'
```

```
> Network id:        11155111 #Red Sepolia Testnet
```

```
> Block gas limit: 30000000 (0x1c9c380)
```

```
2_deploy_community.js
```

```
=====
```

```
Replacing 'Community'
```

```
-----
```

```
> transaction hash:
```

```
0x5bc476df6399dcec3699844531c7a49d46b69894a6adcec6e77c571c317202ecssage.
```

```
> Blocks: 5          Seconds: 52
```

```
> contract address:  0x283bbb7fc922f0d82501e5c4292F4dd3BA068d8d
```

```
> block number:      5810625
```

```
> block timestamp:    1714505832
```

```
> account:            0xc83273f025ecEd0317f52DfE26d95C4638a10D7E
```

```
> balance:            0.234962156593733112
```

```
> gas used:           1601401 (0x186f79)
```

```
> gas price:          2.580861465 gwei
```

```
> value sent:         0 ETH
```

```
> total cost:         0.004132994130912465 ETH
```

```
> Saving artifacts
```

```
-----
```

```
> Total cost:         0.004132994130912465 ETH
```

```
Summary
```

```
=====
```

```
> Total deployments:  1
```

```
> Final cost:         0.004132994130912465 ETH
```

```
ubuntu@ubuntu:~/Documents/GitHub/mi-dao-dapp$ truffle run verify Community
--network dashboard
Verifying contracts on etherscan
  Verifying Community
  Pass - Verified:
https://sepolia.etherscan.io/address/0x283bbb7fc922f0D82501e5c4292F4dd3BA068d8d#code
  Successfully verified 1 contract(s).
Verifying contracts on sourcify
  Verifying Community
  Pass - Verified:
https://sourcify.dev/#/lookup/0x283bbb7fc922f0D82501e5c4292F4dd3BA068d8d
  Successfully verified 1 contract(s).
```

4. Build del proyecto REACT previo a IPFS

```
ubuntu@ubuntu:~/Documents/GitHub/mi-dao-dapp$ npm run build

> mi-dao-dapp@0.1.0 build
> react-scripts build

Creating an optimized production build...
Compiled with warnings.

[eslint]
src/App.1.js
   Line 20:10:    'members' is assigned a value but never used    no-unused-
vars
   Line 117:18:  'handleFetchMembers' is defined but never used    no-unused-
vars

Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.

File sizes after gzip:

  234.74 kB (-1 B)  build/static/js/main.e1e40208.js
   1.78 kB          build/static/js/453.d058d68f.chunk.js
   263 B            build/static/css/main.e6c13ad2.css

The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:

  npm install -g serve
  serve -s build

Find out more about deployment here:
```

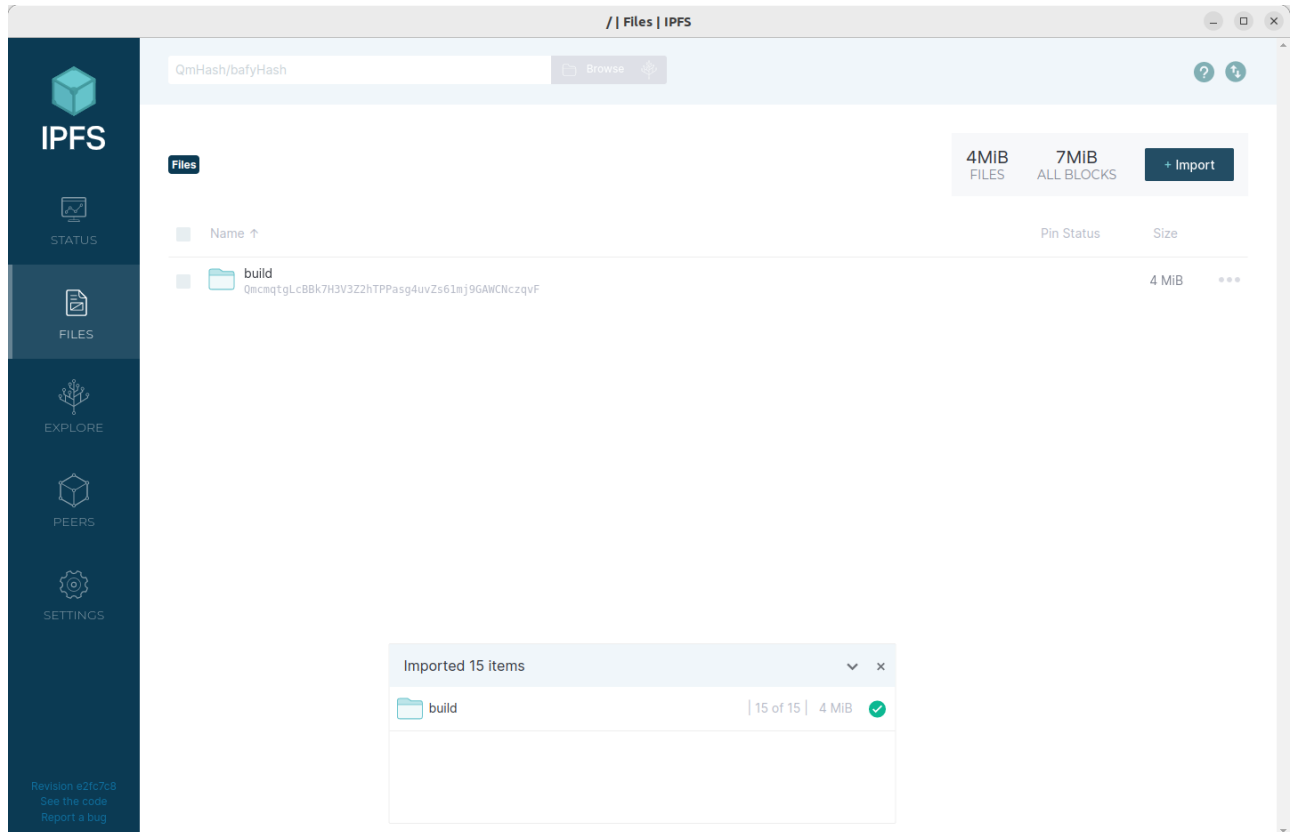
```
https://cra.link/deployment
```

```
npm notice
npm notice New minor version of npm available! 10.5.0 -> 10.6.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.6.0
npm notice Run npm install -g npm@10.6.0 to update!
npm notice
```

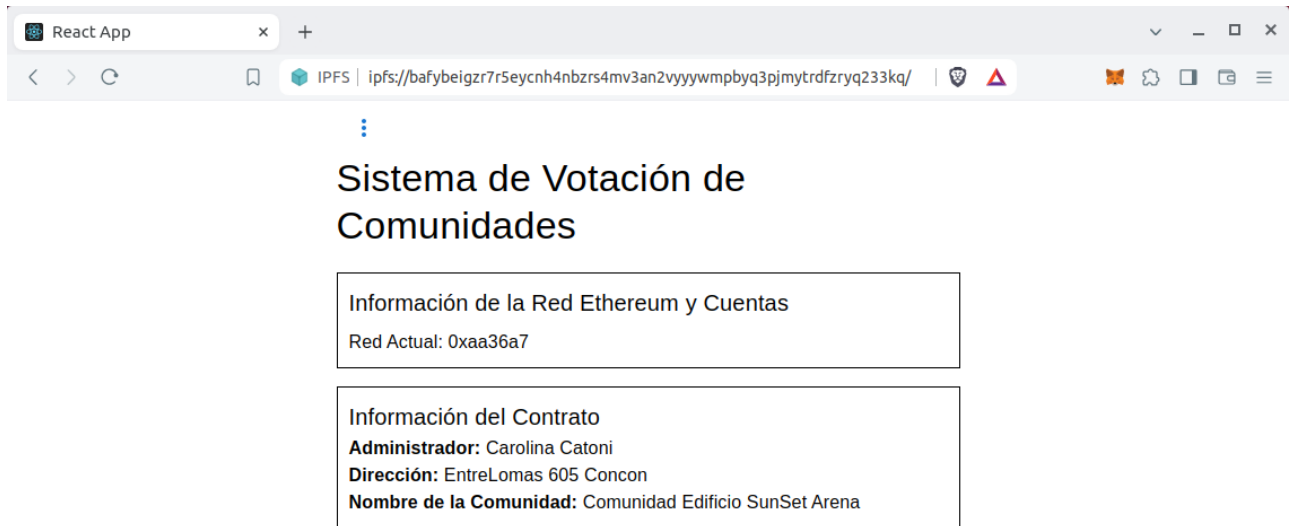
5. Copia y ejecución con IPFS Desktop y Navegador Brave

- **IPFS Dapp Descentralizada:**

ipfs://bafybeigzr7r5eycnh4nbzrs4mv3an2vyywmpbyq3pjmytrdfzryq233kq/



- Ejecucion desde Brave



Instalación de ambiente de desarrollo en Ubuntu 22.04

los pasos para configurar un entorno de desarrollo completo para el desarrollo de aplicaciones descentralizadas (DApps) sobre la blockchain en Ubuntu. La infraestructura creada se puede desglosar en varias partes:

1. Herramientas de desarrollo y versionado: Instalación de Git y GitHub Desktop para manejo de versiones y colaboración en proyectos de código.
2. Editores de código: Instalación de Visual Studio Code (VS Code), un editor de texto optimizado para el desarrollo de software.
3. Entorno de ejecución de JavaScript y gestión de paquetes: Instalación de Node.js (versión LTS) y npm, herramientas esenciales para la ejecución de scripts JavaScript fuera del navegador y la gestión de bibliotecas de JavaScript.
4. Herramientas de desarrollo de contratos inteligentes: Instalación de Truffle y Ganache, que son herramientas para la compilación, migración y prueba de contratos inteligentes en entornos de desarrollo local.
5. Bibliotecas y frameworks para el desarrollo DApp: Instalación de OpenZeppelin (para contratos inteligentes seguros), Material-UI y MUI (para interfaces de usuario en React), y otras librerías necesarias para el diseño y funcionalidad de DApps.
6. Creación de una aplicación React: Utilización de Create React App para establecer la base de una aplicación web que interactuará con la blockchain.
7. Entorno y herramientas para la verificación y despliegue de contratos: Configuración de plugins y herramientas para la verificación de contratos en redes blockchain públicas, gestión de wallets y variables de entorno.
8. Hosting descentralizado: Preparativos para el despliegue de la aplicación en un entorno descentralizado, implicando la construcción de la aplicación para producción.

Esta infraestructura permite el desarrollo, prueba, y despliegue de aplicaciones descentralizadas, integrando herramientas modernas de desarrollo web con tecnologías específicas de blockchain.

Principales versiones de software

- **Folder de proyecto:** `ubuntu@ubuntu:~/Documents/GitHub/DAOCommunity$`
- **truffle** `--version`
- **Truffle v5.11.5 (core: 5.11.5)**
- **Ganache v7.9.1**
- **Solidity - 0.8.20 (solc-js)**
- **Node v20.12.2**
- **Web3.js v1.10.0**

Instrucciones para recrear el ambiente

```
# Install Github Desktop & Git
sudo apt-get update
sudo apt-get install git

wget -qO - https://apt.packages.shiftkey.dev/gpg.key | gpg --dearmor | sudo
tee /usr/share/keyrings/shiftkey-packages.gpg > /dev/null
sudo sh -c 'echo "deb [arch=amd64 signed-by=/usr/share/keyrings/shiftkey-
packages.gpg] https://apt.packages.shiftkey.dev/ubuntu/ any main" >
/etc/apt/sources.list.d/shiftkey-packages.list'
sudo apt update && sudo apt install github-desktop

# Install VS Code
sudo apt-get install wget gpg
wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor
> packages.microsoft.gpg
sudo install -D -o root -g root -m 644 packages.microsoft.gpg
/etc/apt/keyrings/packages.microsoft.gpg
sudo sh -c 'echo "deb [arch=amd64,arm64,armhf signed-
by=/etc/apt/keyrings/packages.microsoft.gpg]
https://packages.microsoft.com/repos/code stable main" >
/etc/apt/sources.list.d/vscode.list'
rm -f packages.microsoft.gpg
sudo apt install apt-transport-https
sudo apt update
sudo apt install code

# Install nodejs LTS
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install curl
sudo apt-get update
curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -
sudo apt-get install nodejs
node -v
npm -v

#Install truffle
sudo npm install -g truffle
sudo npm install -g truffle dashboard
```

```
truffle --version

#Install Ganche
wget https://github.com/trufflesuite/ganache-
ui/releases/download/v2.7.1/ganache-2.7.1-linux-x86_64.AppImage
sudo add-apt-repository universe
sudo apt install libfuse2

# Install library : OpenZeppelin; MAterial UI: otras
npm install @openzeppelin/contracts
npm install @material-ui/core
npm install @mui/material @emotion/react @emotion/styled //import
MoreVertIcon from '@mui/icons-material/MoreVert'
npm install @mui/icons-material

npm install react-error-boundary
npm install web3

# Install react en proyecto
npx create-react-app mi-dao-dapp

#Install pluggins verufy
npm install @truffle/dashboard
npm install -D truffle-plugin-verify
npm install @truffle/hdwallet-provider
truffle run verify <ContractName> --network <NetworkName>

#Creación de fichero de variables de entorno
touch .env
npm install dotenv

# Hosting Descentralizado
npm run build
```

Smart Contract

Smart Contract desde Truffle dashboard a Sepolia TestNet

- **El contrato está desplegado en la dirección:**<https://sepolia.etherscan.io/address/0x5b244ddac206f403430faf50e950b8cfabc8ec3b>

Funcionalidades del Contrato

- **Registro de miembros:** Los miembros pueden registrarse almacenando una cantidad de tokens que les permitirá votar.
- **Creación de propuestas:** Los administradores pueden crear propuestas sobre las cuales los miembros podrán votar.
- **Votación:** Los miembros utilizan sus tokens para votar en propuestas activas.

- **Ejecución de propuestas:** Las propuestas que alcanzan los votos necesarios pueden ser ejecutadas por los administradores.

Métodos Principales y componentes de despliegue y validación

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

// Contrato de la Comunidad
contract Community is Ownable(msg.sender) {
    string public communityName;
    string public communityAddress;
    string public presidentName;
    string public adminName;
    uint256 public creationDate;

    mapping(address => uint256) public memberTokens; // Tokens de votación
    asignados a cada miembro
    Proposal[] public proposals; // Lista de propuestas
    mapping(uint256 => mapping(address => bool)) public hasVoted; //
    Mapping to track if a member has voted on a proposal

    struct Proposal {
        string description;
        uint256 voteCount;
        bool executed;
    }

    event ProposalCreated(uint256 proposalId, string description);
    event VoteReceived(uint256 proposalId, address voter, uint256 votes);
    event ProposalExecuted(uint256 proposalId);

    constructor(
        string memory _name,
        string memory _address,
        string memory _president,
        string memory _admin,
        uint256 _creationDate
    ) {
        communityName = _name;
        communityAddress = _address;
        presidentName = _president;
        adminName = _admin;
        creationDate = _creationDate;
    }

    function registerMember(address member, uint256 tokens) public
    onlyOwner {
        memberTokens[member] = tokens;
    }
}
```

```
}

function createProposal(string memory description) public onlyOwner {
    proposals.push(Proposal({
        description: description,
        voteCount: 0,
        executed: false
    }));

    uint256 newProposalId = proposals.length - 1;
    emit ProposalCreated(newProposalId, description);
}

function voteOnProposal(uint256 proposalId, uint256 votes) public {
    require(memberTokens[msg.sender] >= votes, "Not enough tokens to
vote");
    require(!hasVoted[proposalId][msg.sender], "Already voted");

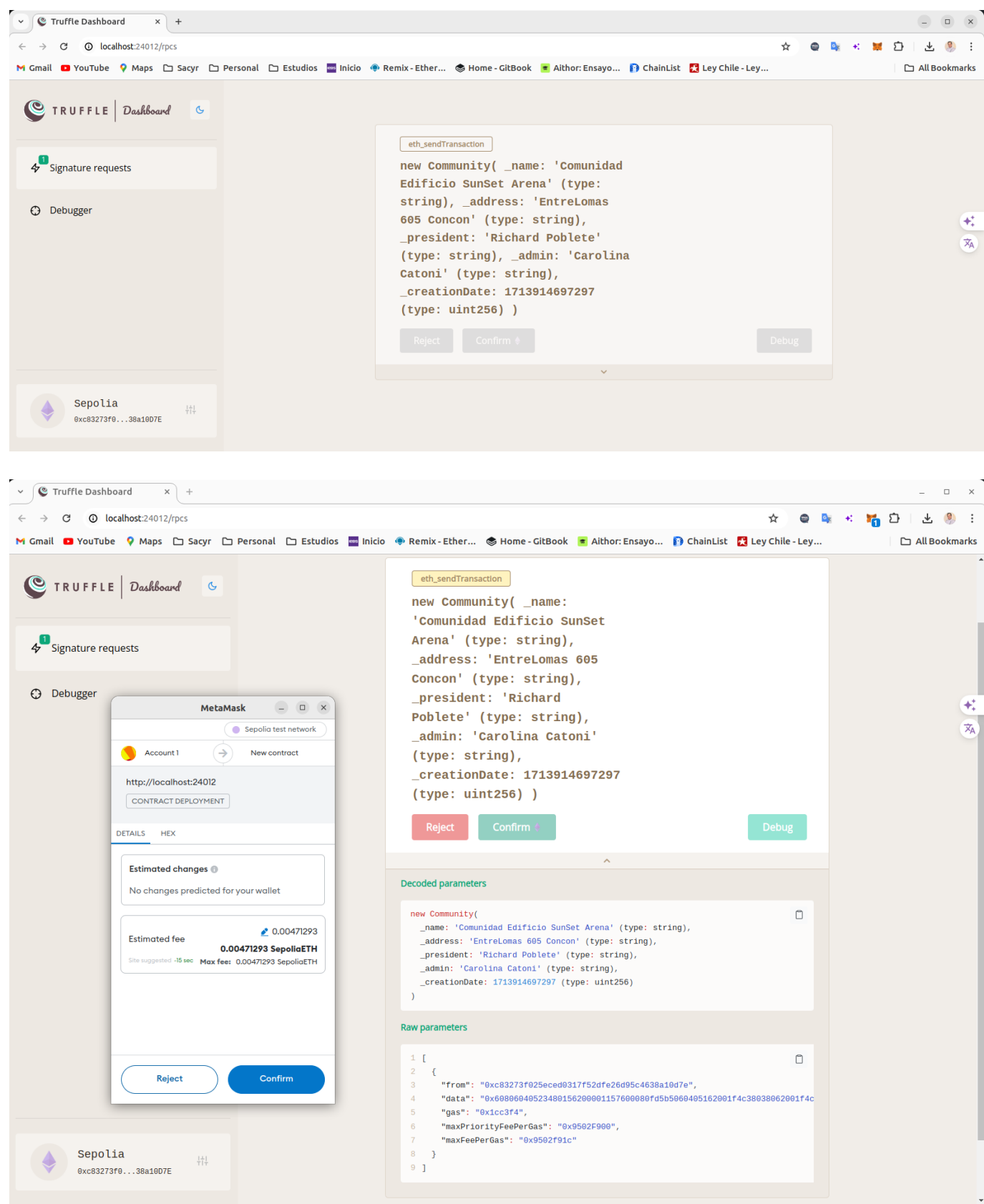
    proposals[proposalId].voteCount += votes;
    hasVoted[proposalId][msg.sender] = true;
    memberTokens[msg.sender] -= votes;

    emit VoteReceived(proposalId, msg.sender, votes);
}

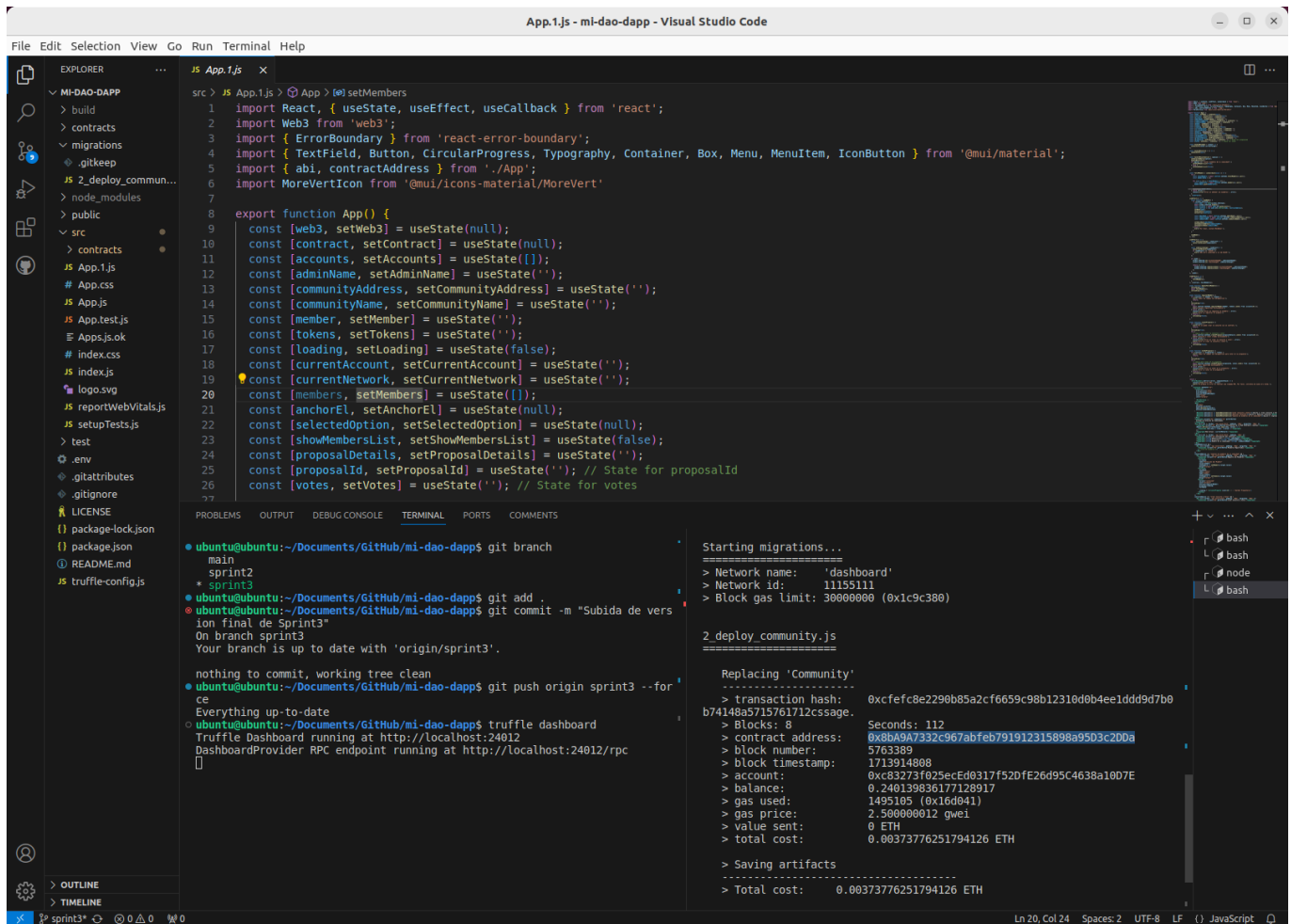
function executeProposal(uint256 proposalId) public onlyOwner {
    require(!proposals[proposalId].executed, "Proposal already
executed");
    proposals[proposalId].executed = true;
    // Logic to execute the proposal goes here

    emit ProposalExecuted(proposalId);
}
}
```

Ganache Dashboard



Despliegue desde VS Code a Sepolia TestNet



Contrato de la Comunidad

Descripción General

Este contrato inteligente, desarrollado con Solidity, permite la gestión de una comunidad mediante la asignación de tokens de votación a sus miembros y la creación y votación de propuestas. Está diseñado para ser administrado por el propietario del contrato, quien tiene derechos exclusivos para ciertas operaciones.

Características del Contrato

- **Gestión de Miembros y Propuestas:** Solo el propietario puede añadir miembros y crear propuestas.
- **Votación:** Los miembros pueden votar en propuestas activas utilizando tokens asignados.

Detalles del Contrato

Importaciones

- **OpenZeppelin Contracts:** Utiliza **Ownable** para funciones de propiedad y **ERC20** para manejar tokens.

```
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

Variables de Estado

communityName, communityAddress, presidentName, adminName, creationDate: Almacenan información básica sobre la comunidad. **memberTokens:** Un mapping que relaciona cada dirección de miembro con sus tokens de votación. **proposals:** Un array que almacena todas las propuestas creadas. **hasVoted:** Un mapping doble para rastrear si un miembro ha votado en una propuesta.

Structs

Proposal: Define una propuesta con descripción, conteo de votos y estado de ejecución.

```
struct Proposal {
    string description;
    uint256 voteCount;
    bool executed;
}
```

Eventos

ProposalCreated: Se emite cuando se crea una nueva propuesta. **VoteReceived:** Se emite cuando un voto es registrado. **ProposalExecuted:** Se emite cuando una propuesta es ejecutada. ****Constructor Inicializa el contrato con datos básicos de la comunidad y establece el propietario.

```
constructor(string memory _name, string memory _address, string memory
_president, string memory _admin, uint256 _creationDate) Ownable() {
    communityName = _name;
    communityAddress = _address;
    presidentName = _president;
    adminName = _admin;
    creationDate = _creationDate;
}
```

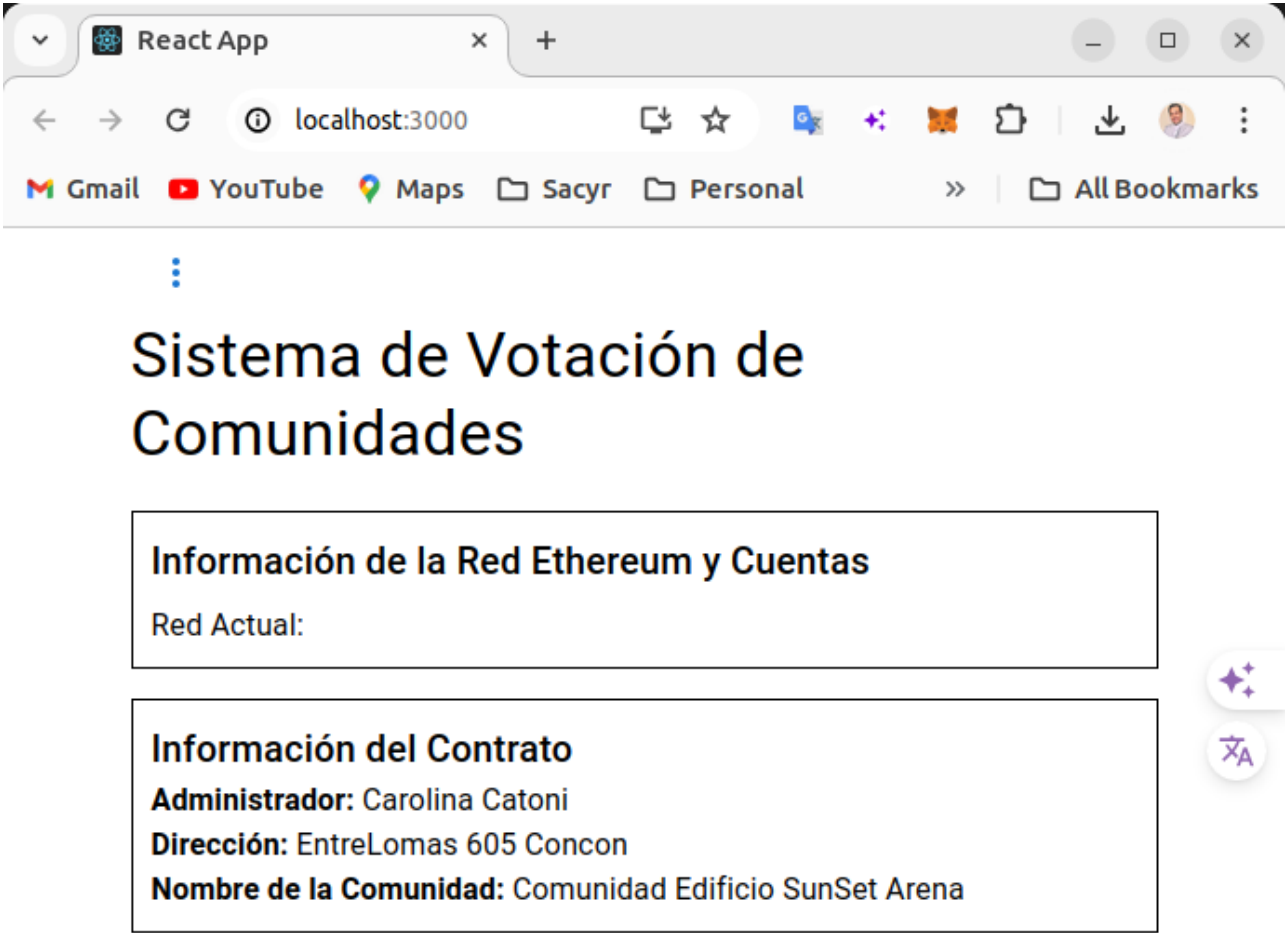
Funciones del Contrato, pantallas y pasos

registerMember: Asigna tokens a un nuevo miembro. Solo puede ser llamada por el propietario. **createProposal:** Crea una nueva propuesta. Solo puede ser iniciada por el propietario. **voteOnProposal:** Permite a los miembros votar en propuestas activas. **executeProposal:** Ejecuta una propuesta aprobada. Solo el propietario puede ejecutarla.

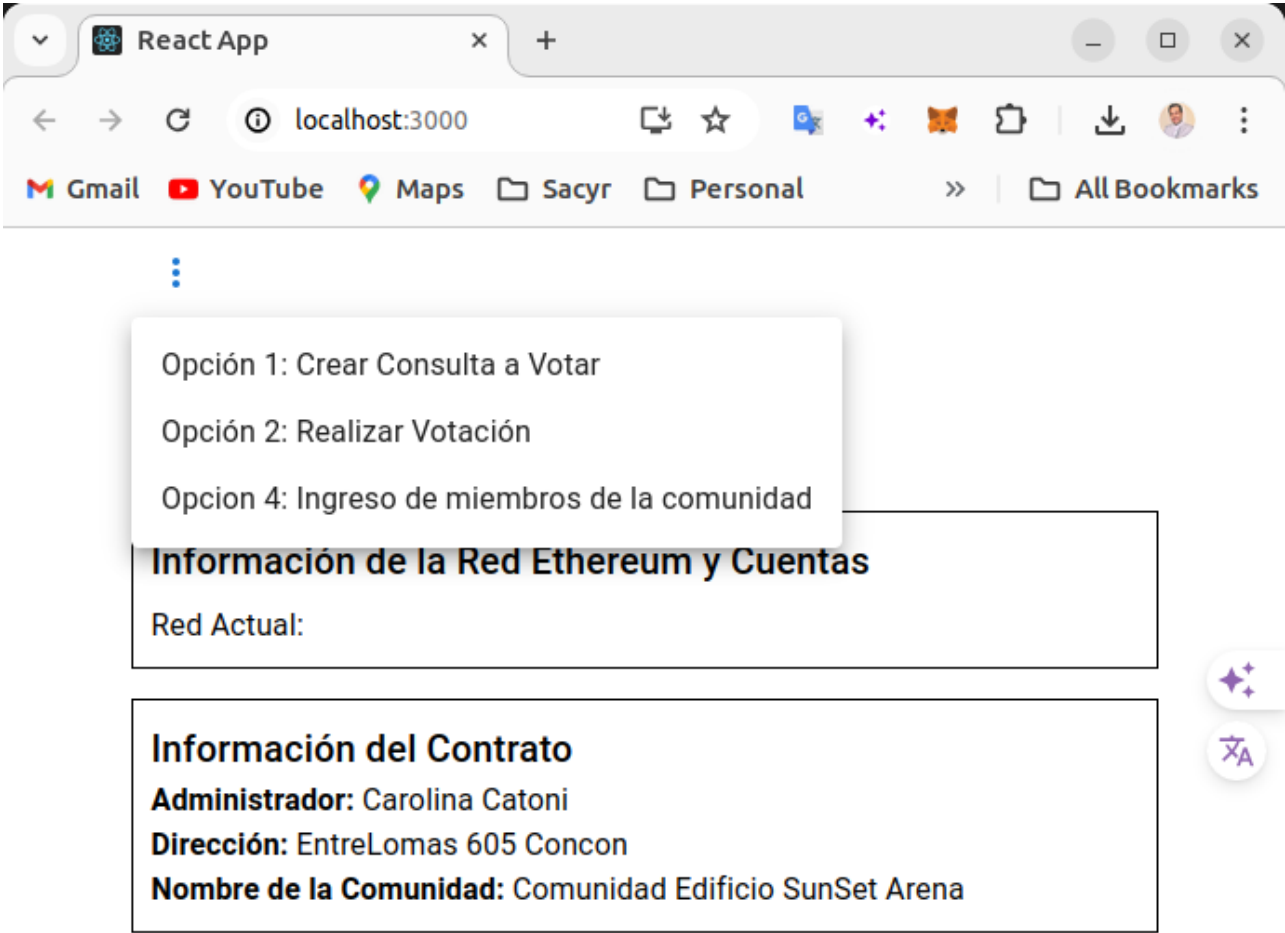
Frontend

El frontend está desarrollado en React y utiliza Web3.js para interactuar con el contrato inteligente a través de MetaMask.

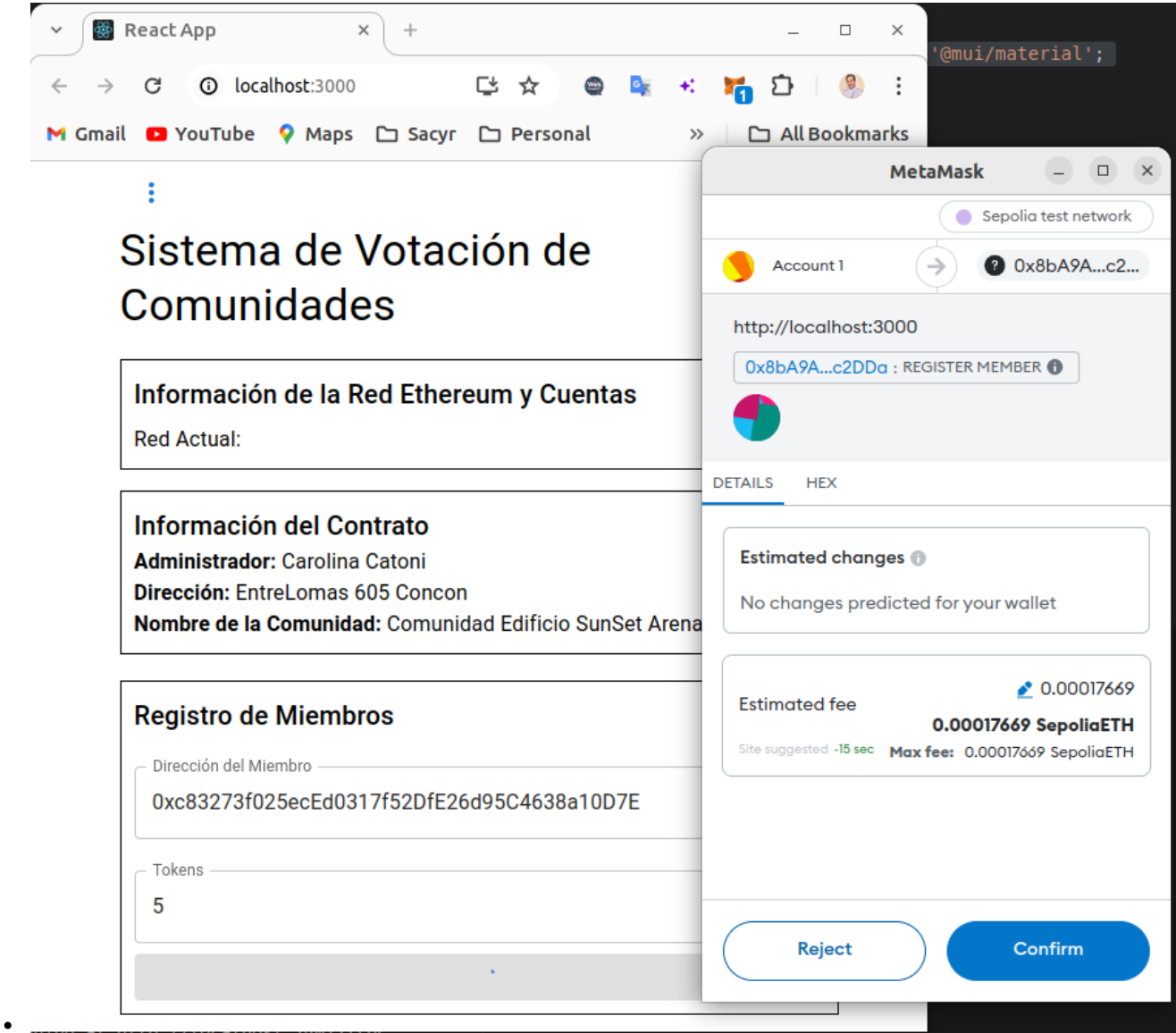
- **Pagina Inicio**

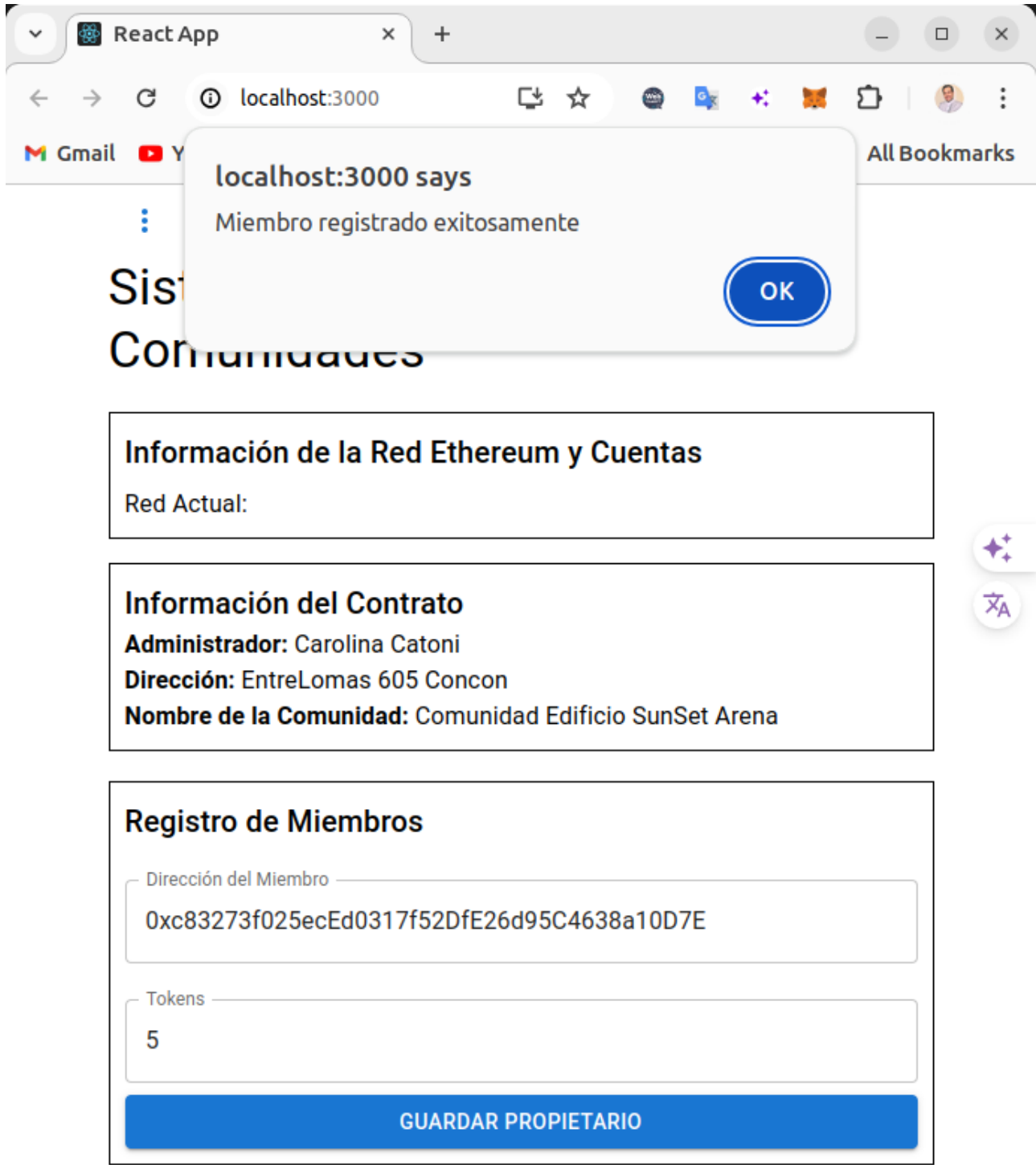


-
- **Menú**

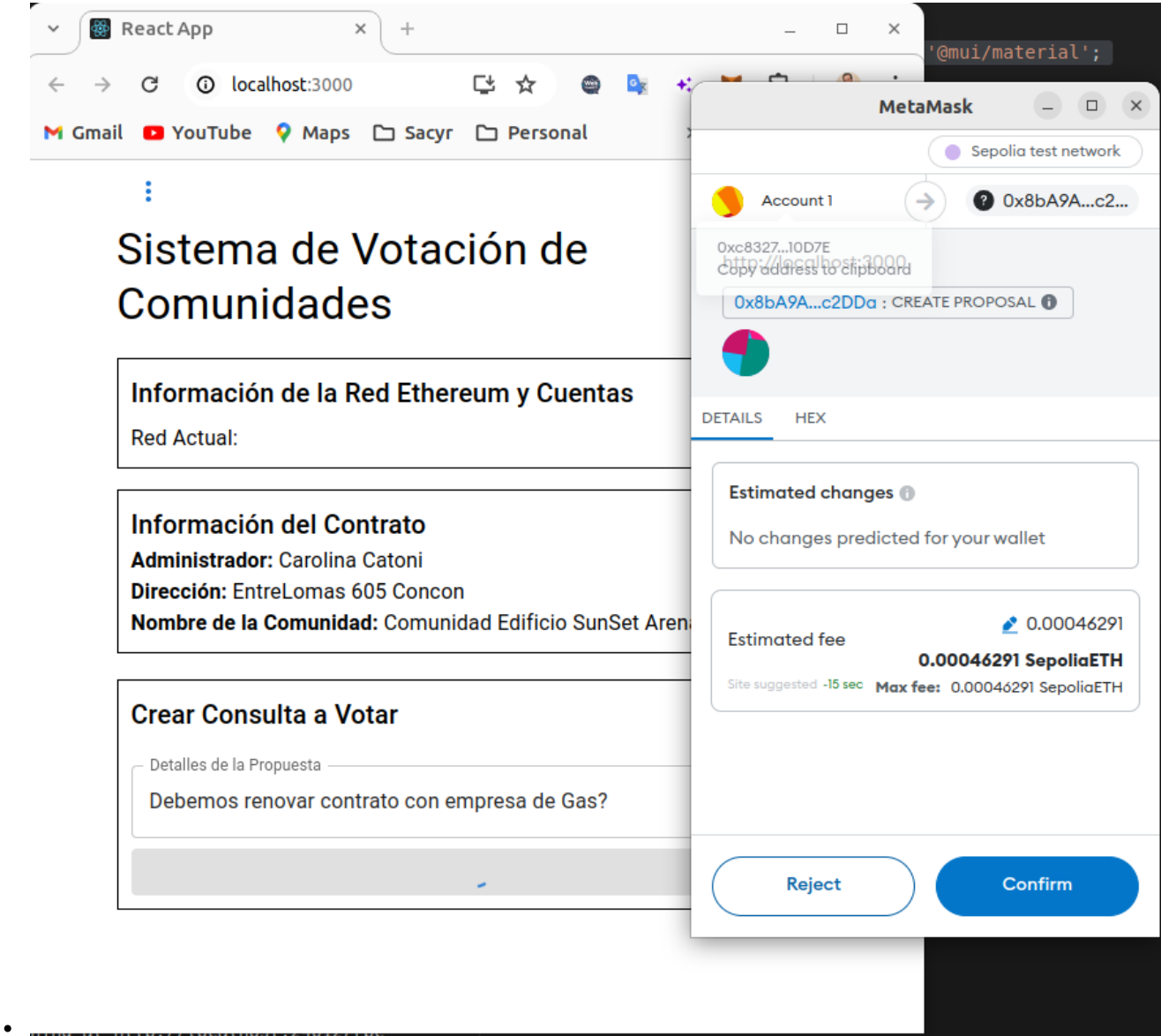


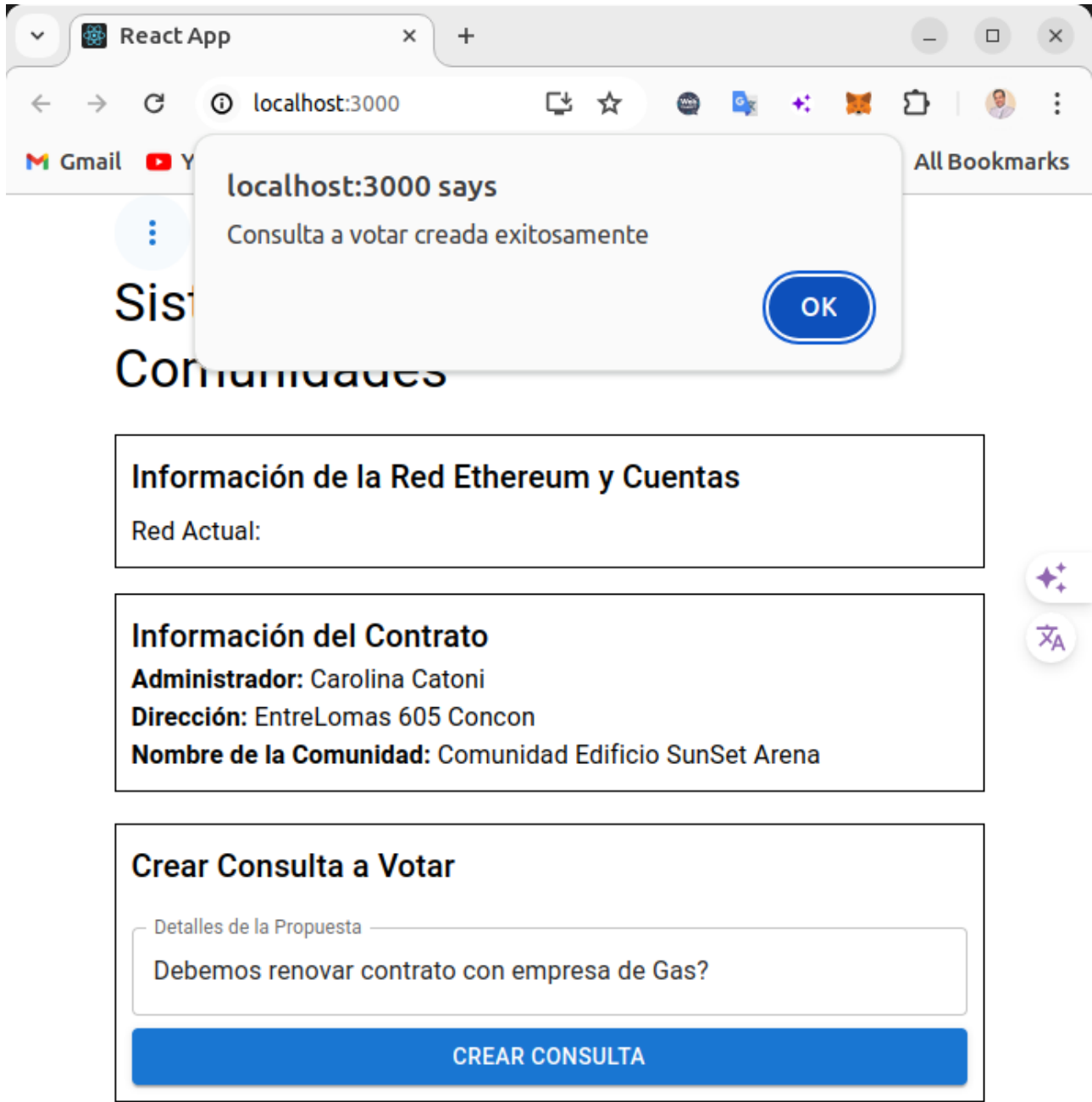
-
- Incorporación de un miembro para votar



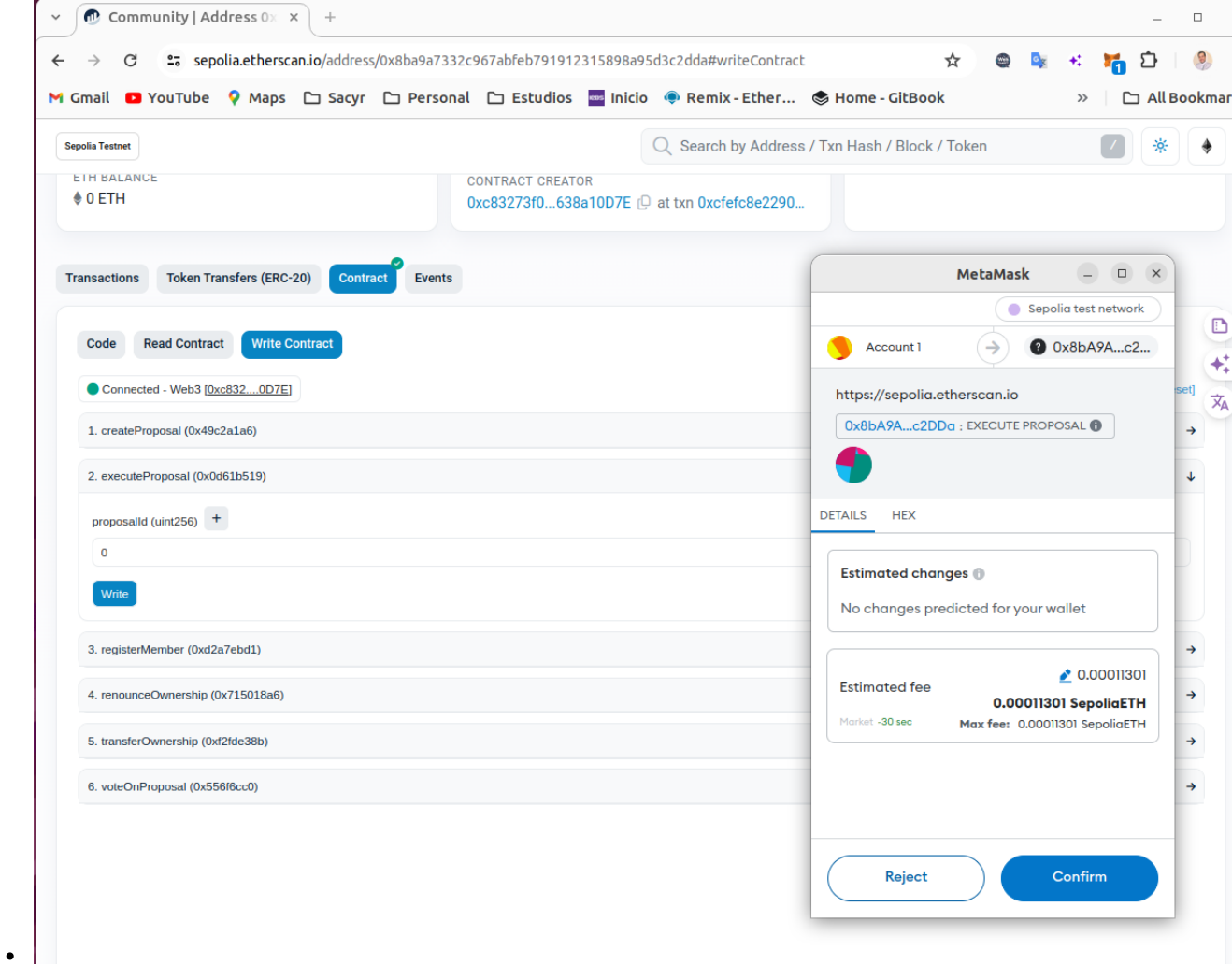


-
- Creación de votación

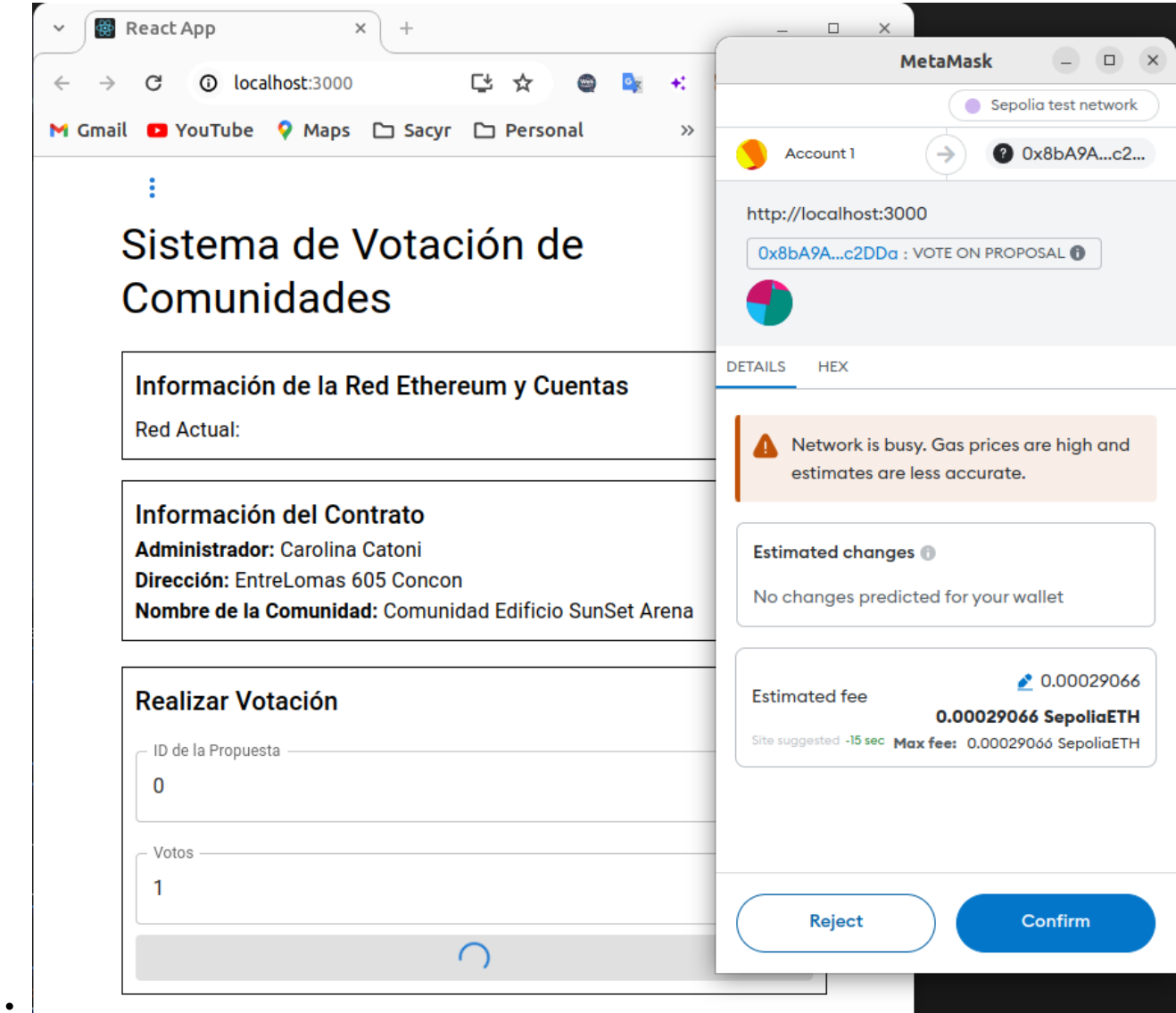


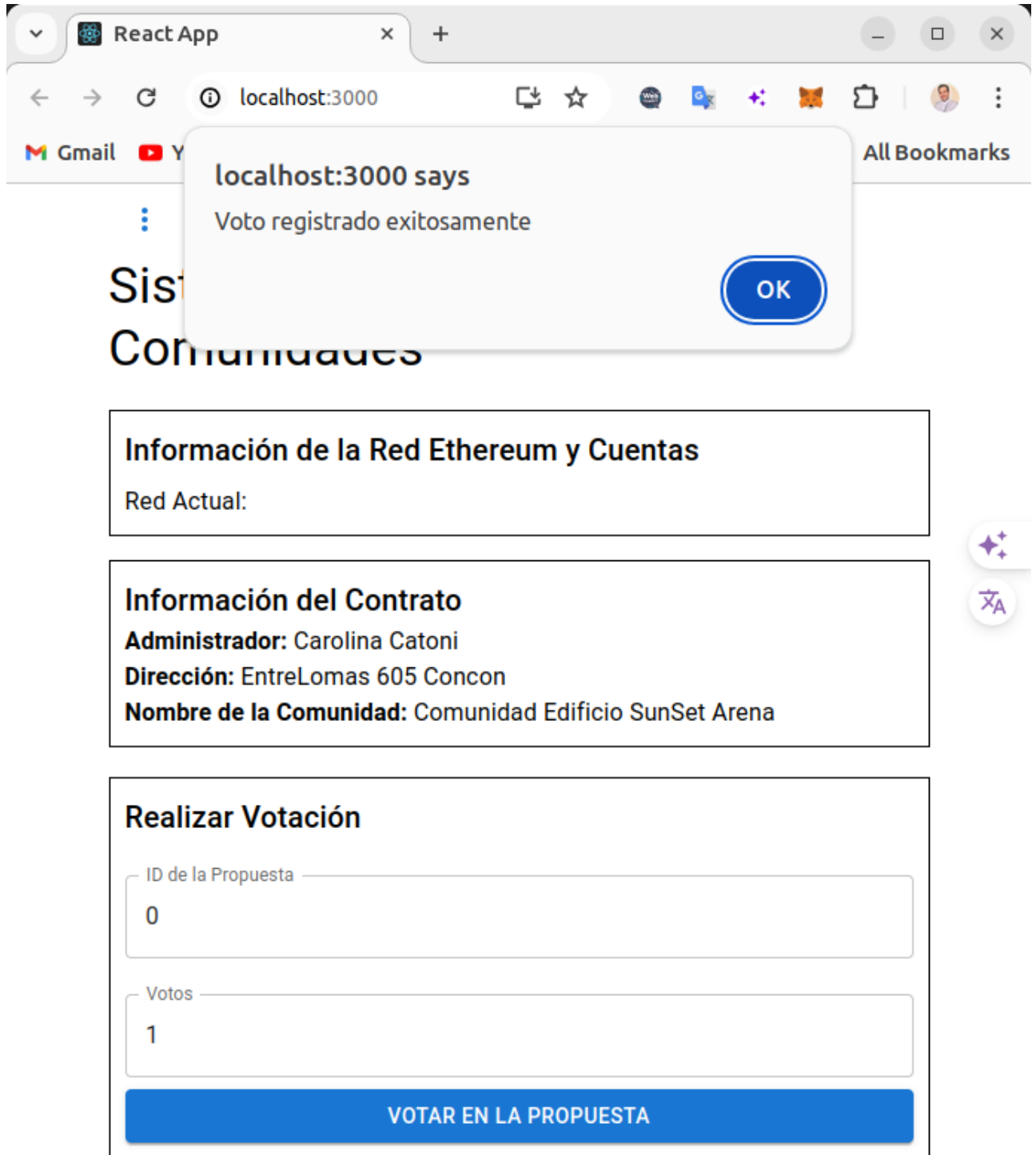


-
- **Habilitación Manual de Proposal**



- **Votación**





Componentes Principales

Formulario de Registro de Miembro: Permite a los usuarios registrarse como miembros. Creador de

Propuestas: Interfaz para que los administradores creen nuevas propuestas. Votaciones: Permite a los miembros votar en propuestas activas.

Fuentes de los Componentes

```
import React, { useState, useEffect, useCallback } from 'react';
import Web3 from 'web3';
import { ErrorBoundary } from 'react-error-boundary';
import { TextField, Button, CircularProgress, Typography, Container, Box,
```

```
Menu, MenuItem, IconButton } from '@mui/material';
import { abi, contractAddress } from './App';
import MoreVertIcon from '@mui/icons-material/MoreVert'

export function App() {
  const [web3, setWeb3] = useState(null);
  const [contract, setContract] = useState(null);
  const [accounts, setAccounts] = useState([]);
  const [adminName, setAdminName] = useState('');
  const [communityAddress, setCommunityAddress] = useState('');
  const [communityName, setCommunityName] = useState('');
  const [member, setMember] = useState('');
  const [tokens, setTokens] = useState('');
  const [loading, setLoading] = useState(false);
  const [currentAccount, setCurrentAccount] = useState('');
  const [currentNetwork, setCurrentNetwork] = useState('');
  const [members, setMembers] = useState([]);
  const [anchorEl, setAnchorEl] = useState(null);
  const [selectedOption, setSelectedOption] = useState(null);
  const [showMembersList, setShowMembersList] = useState(false);
  const [proposalDetails, setProposalDetails] = useState('');
  const [proposalId, setProposalId] = useState(''); // State for proposalId
  const [votes, setVotes] = useState(''); // State for votes

  const handleMenuOpen = (event) => {
    setAnchorEl(event.currentTarget);
  };

  const handleMenuClose = () => {
    setAnchorEl(null);
  };

  const handleMenuItemClick = (option) => {
    setSelectedOption(option);
    handleMenuClose();
    if (option === "Listar miembros de la comunidad") {
      setShowMembersList(true);
    } else {
      setShowMembersList(false);
    }
  };

  const fetchMembers = useCallback(async () => {
    try {
      const totalMembers = await contract.methods.totalMembers().call();
      const membersData = [];

      for (let i = 0; i < totalMembers; i++) {
        const memberInfo = await contract.methods.members(i).call();
        membersData.push(memberInfo);
      }

      setMembers(membersData);
    } catch (error) {
```

```
        console.error('Error al obtener los miembros:', error);
    }
}, [contract]);

useEffect(() => {
    async function loadWeb3() {
        if (window.ethereum) {
            const web3 = new Web3(window.ethereum);
            await window.ethereum.enable();
            const accounts = await web3.eth.getAccounts();
            const contract = new web3.eth.Contract(abi, contractAddress);
            setWeb3(web3);
            setAccounts(accounts);
            setContract(contract);

            const adminName = await contract.methods.adminName().call();
            const communityAddr = await
contract.methods.communityAddress().call();
            const communityNm = await contract.methods.communityName().call();

            setAdminName(adminName);
            setCommunityAddress(communityAddr);
            setCommunityName(communityNm);
        } else {
            alert('Por favor, instale MetaMask!');
        }
    }

    loadWeb3();
}, []);

useEffect(() => {
    const onAccountChange = newAccount => {
        setCurrentAccount(newAccount);
    };

    const onNetworkChange = newNetwork => {
        setCurrentNetwork(newNetwork);
        if (newNetwork !== '0x539') {
            alert('¡No estás conectado a la red 0x539!');
        }
    };

    if (web3) {
        window.ethereum.on('accountsChanged', onAccountChange);
        window.ethereum.on('chainChanged', onNetworkChange);

        return () => {
            window.ethereum.removeListener('accountsChanged', onAccountChange);
            window.ethereum.removeListener('chainChanged', onNetworkChange);
        };
    }
}, [web3]);
```



```
useEffect(() => {
  if (contract) {
    fetchMembers();
  }
}, [contract, fetchMembers]);

async function handleFetchMembers() {
  setLoading(true);
  await fetchMembers();
  setLoading(false);
}

async function registerMember() {
  if (!contract || !member || !tokens) {
    alert('Todos los campos son obligatorios');
    return;
  }
  setLoading(true);
  try {
    await contract.methods.registerMember(member, tokens).send({ from:
accounts[0] });
    alert('Miembro registrado exitosamente');
  } catch (error) {
    console.error('Error al registrar el miembro:', error);
    alert('Error al registrar el miembro');
  } finally {
    setLoading(false);
  }
}

async function createProposal() {
  if (!contract) {
    alert('No se puede crear la consulta sin un contrato.');
```

```
    return;
  }
  setLoading(true);
  try {
    // Lógica para crear la consulta a votar
    await contract.methods.createProposal(proposalDetails).send({ from:
accounts[0] });
    alert('Consulta a votar creada exitosamente');
  } catch (error) {
    console.error('Error al crear la consulta a votar:', error);
    alert('Error al crear la consulta a votar');
  } finally {
    setLoading(false);
  }
}

async function voteOnProposal() {
  if (!contract || !proposalId || !votes) {
    alert('Todos los campos son obligatorios para votar en la
propuesta');
```

```
    return;
  }
}
```

```

    }
    setLoading(true);
    try {
      // Lógica para votar en la propuesta
      await contract.methods.voteOnProposal(proposalId, votes).send({ from:
accounts[0] });
      alert('Voto registrado exitosamente');
    } catch (error) {
      console.error('Error al votar en la propuesta:', error);
      alert('Error al votar en la propuesta');
    } finally {
      setLoading(false);
    }
  }

  return (
    <ErrorBoundary onError={(error, componentStack) => {
      console.error('Error RPC:', error);
      alert('Se produjo un error al realizar una llamada RPC. Por favor,
inténtelo de nuevo más tarde.')}>
      <Container maxWidth="sm">
        <IconButton
          aria-controls="menu"
          aria-haspopup="true"
          onClick={handleMenuOpen}
          size="large"
          color="primary"
        >
          <MoreVertIcon />
        </IconButton>
        <Menu
          id="menu"
          anchorEl={anchorEl}
          open={Boolean(anchorEl)}
          onClose={handleMenuClose}
        >
          <MenuItem onClick={() => handleMenuItemClick("Crear Consulta a
Votar")}>Opción 1: Crear Consulta a Votar</MenuItem>
          <MenuItem onClick={() => handleMenuItemClick("Realizar
Votación")}>Opción 2: Realizar Votación</MenuItem>
          <MenuItem onClick={() => handleMenuItemClick("Ingreso de miembros
de la comunidad")}>Opcion 4: Ingreso de miembros de la comunidad</MenuItem>
        </Menu>
        <Typography variant="h4" component="h1" gutterBottom>
          Sistema de Votación de Comunidades
        </Typography>
        <Box sx={{ mb: 2, border: '1px solid black', padding: '10px',
marginTop: '20px' }}>
          <Typography variant="h6" gutterBottom>Información de la Red
Ethereum y Cuentas</Typography>
          {Object.entries(currentAccount).map(([key, value]) => (
            <Typography key={key}>`${key}: ${value}`</Typography>
          ))}
        </Box>
      </Container>
    </ErrorBoundary>
  );
}

```

```

        <Typography>Red Actual: {currentNetwork}</Typography>
      </Box>
      <Box sx={{ mb: 2, border: '1px solid black', padding: '10px' }}>
        <Typography variant="h6">Información del Contrato</Typography>
        <Typography><strong>Administrador:</strong> {adminName}
      </Typography>
        <Typography><strong>Dirección:</strong> {communityAddress}
      </Typography>
        <Typography><strong>Nombre de la Comunidad:</strong>
      {communityName}</Typography>
      </Box>
      {showMembersList && (
        <Box sx={{ border: '1px solid black', padding: '10px', marginTop:
'20px' }}>
          <Typography variant="h6" gutterBottom>Miembros
Registrados</Typography>
          {/* Lista de miembros */}
        </Box>
      )}
      {selectedOption === "Ingreso de miembros de la comunidad" && (
        <Box sx={{ border: '1px solid black', padding: '10px', marginTop:
'20px' }}>
          <Typography variant="h6" gutterBottom>Registro de
Miembros</Typography>
          <TextField
            fullWidth
            label="Dirección del Miembro"
            value={member}
            onChange={e => setMember(e.target.value)}
            margin="normal" />
          <TextField
            fullWidth
            label="Tokens"
            type="number"
            value={tokens}
            onChange={e => setTokens(e.target.value)}
            margin="normal" />
          <Button
            variant="contained"
            color="primary"
            onClick={registerMember}
            disabled={loading}
            fullWidth
          >
            {loading ? <CircularProgress size={24} /> : 'Guardar
Propietario'}
          </Button>
        </Box>
      )}
      {selectedOption === "Crear Consulta a Votar" && (
        <Box sx={{ border: '1px solid black', padding: '10px', marginTop:
'20px' }}>
          <Typography variant="h6" gutterBottom>Crear Consulta a
Votar</Typography>

```

```

        <TextField
          fullWidth
          label="Detalles de la Propuesta"
          value={proposalDetails}
          onChange={e => setProposalDetails(e.target.value)}
          margin="normal" />
        <Button
          variant="contained"
          color="primary"
          onClick={createProposal}
          disabled={loading}
          fullWidth
        >
          {loading ? <CircularProgress size={24} /> : 'Crear Consulta'}
        </Button>
      </Box>
    )}
    {selectedOption === "Realizar Votación" && (
      <Box sx={{ border: '1px solid black', padding: '10px', marginTop:
'20px' }}>
        <Typography variant="h6" gutterBottom>Realizar
Votación</Typography>
        <TextField
          fullWidth
          label="ID de la Propuesta"
          type="number"
          value={proposalId}
          onChange={e => setProposalId(e.target.value)}
          margin="normal" />
        <TextField
          fullWidth
          label="Votos"
          type="number"
          value={votes}
          onChange={e => setVotes(e.target.value)}
          margin="normal" />
        <Button
          variant="contained"
          color="primary"
          onClick={voteOnProposal}
          disabled={loading}
          fullWidth
        >
          {loading ? <CircularProgress size={24} /> : 'Votar en la
Propuesta'}
        </Button>
      </Box>
    )}
  </Container>
</ErrorBoundary>
);
}

```

Interacción con el Contrato de la Comunidad

Este proyecto React permite interactuar con un contrato inteligente de Ethereum para gestionar una comunidad. Los usuarios pueden registrar miembros y consultar información del contrato directamente desde la interfaz.

Funcionalidades

- **Conexión con MetaMask:** Conexión automática a MetaMask para interactuar con la blockchain.
- **Registro de miembros:** Permite a los administradores registrar nuevos miembros y asignarles tokens.
- **Visualización de información del contrato:** Muestra datos como el nombre del administrador, la dirección y el nombre de la comunidad.

Tecnologías Utilizadas

- React.js
- Web3.js
- Material-UI: para los componentes de la interfaz de usuario.
- Ethereum Blockchain

Instalación y Configuración

Asegúrate de tener instalado `Node.js` y `npm`. Sigue los pasos para ejecutar la aplicación:

1. Clona el repositorio.
2. Instala las dependencias necesarias:

Detalles del Código

Componente `App`

Es el componente principal que maneja la conexión con Ethereum y muestra la interfaz de usuario.

Variables de Estado

- `web3`: Instancia de Web3 usada en toda la aplicación.
- `contract`: Instancia del contrato inteligente.
- `accounts`: Lista de cuentas disponibles desde el wallet.
- `adminName`, `communityAddress`, `communityName`: Variables para almacenar y mostrar la información del contrato.
- `member`: Dirección del nuevo miembro a registrar.
- `tokens`: Número de tokens a asignar al nuevo miembro.
- `loading`: Estado para controlar la visualización del indicador de carga durante las transacciones.

Hook `useEffect` de ejemplo

Inicializa la conexión con MetaMask y carga los datos iniciales desde el contrato:

```

useEffect(() => {
  async function loadWeb3() {
    if (window.ethereum) {
      const web3 = new Web3(window.ethereum);
      await window.ethereum.enable();
      const accounts = await web3.eth.getAccounts();
      const contract = new web3.eth.Contract(abi, contractAddress);
      setWeb3(web3);
      setAccounts(accounts);
      setContract(contract);

      const adminName = await contract.methods.adminName().call();
      const communityAddr = await contract.methods.communityAddress().call();
      const communityNm = await contract.methods.communityName().call();

      setAdminName(adminName);
      setCommunityAddress(communityAddr);
      setCommunityName(communityNm);
    } else {
      alert(';Instala MetaMask!');
    }
  }

  loadWeb3();
}, []);

```

Función registerMember

Maneja el registro de un nuevo miembro enviando una transacción al contrato inteligente:

```

const registerMember = async () => {
  if (!contract || !member || !tokens) {
    alert('Todos los campos son obligatorios');
    return;
  }
  setLoading(true);
  try {
    await contract.methods.registerMember(member, tokens).send({ from:
accounts[0] });
    alert('Miembro registrado con éxito');
  } catch (error) {
    console.error('Error al registrar miembro:', error);
    alert('Falló el registro del miembro');
  } finally {
    setLoading(false);
  }
};

```

Mejoras Futuras

Manejo avanzado de errores: Proporcionar retroalimentación más detallada sobre los errores de blockchain. OK **Actualizaciones en tiempo real:** Implementar WebSocket u otra tecnología para obtener actualizaciones en tiempo real del estado del contrato. **Accesibilidad mejorada:** Asegurar que la interfaz cumpla con los estándares de accesibilidad. **Completar otras funciones del contrato:** Pendientes funcionalidades del Smart Contrat y mejoras en la usabilidad de cara al usuario.

Licencia

Este proyecto está bajo la Licencia MIT.