import React, { useState, useEffect, useCallback } from 'react';

import Web3 from 'web3';

import './App.css';

import contractABI from './HealthDataSharing.json';

const contractAddress = '0x5FbDB2315678afecb367f032d93F642f64180aa3'; // Replace with your contract address

const App = () => {

const [account, setAccount] = useState('');

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

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

const [healthcareExperts, setHealthcareExperts] = useState([]);

const [patients, setPatients] = useState([]);

const [notifications, setNotifications] = useState([]);

const [expertAddress, setExpertAddress] = useState('');

const [healthData, setHealthData] = useState('');

const [message, setMessage] = useState('');

const [patientAddress, setPatientAddress] = useState('');

const [loading, setLoading] = useState(false);

const [familyMemberAddress, setFamilyMemberAddress] = useState('');

const [consentToRI, setConsentToRI] = useState(false);

const [rewardAmount, setRewardAmount] = useState(0);

const [researchInstitutes, setResearchInstitutes] = useState([]);

const [familyMembers, setFamilyMembers] = useState([]);

const [familyMemberName, setFamilyMemberName] = useState('');

// Initialize Web3 and Contract

const initWeb3 = useCallback(async () => {

if (window.ethereum) {

try {

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

const web3Instance = new Web3(window.ethereum);

setWeb3(web3Instance);

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

setAccount(accounts[0]);

const contractInstance = new web3Instance.eth.Contract(contractABI, contractAddress);

setContract(contractInstance);

console.log("Contract initialized:", contractInstance);

} catch (error) {

console.error('Failed to load web3 or accounts:', error);

alert('Failed to connect to MetaMask. Please make sure it is installed and unlocked.');

}

} else {

alert('Please install MetaMask to use this dApp!');

}

}, []);

useEffect(() => {

initWeb3();

}, [initWeb3]);

useEffect(() => {

if (window.ethereum) {

window.ethereum.on('accountsChanged', (accounts) => setAccount(accounts[0]));

window.ethereum.on('chainChanged', () => window.location.reload());

}

return () => {

if (window.ethereum) {

window.ethereum.removeAllListeners();

}

};

}, []);

useEffect(() => {

console.log("Contract state changed:", contract);

fetchData(); // Fetch data whenever contract changes

}, [contract]);

// Fetch all necessary data from the blockchain

const fetchData = useCallback(async () => {

if (contract && account) {

setLoading(true);

try {

const experts = await contract.methods.getHealthcareExpertAddresses().call();

setHealthcareExperts(experts);

const patientList = await contract.methods.getPatientAddresses().call();

setPatients(patientList);

const notifs = await contract.methods.viewNotifications().call({ from: account });

setNotifications(notifs);

const family = await contract.methods.getFamilyMembers().call();

setFamilyMembers(family);

const research = await contract.methods.getResearchInstituteAddresses().call();

setResearchInstitutes(research);

} catch (error) {

console.error('Error fetching data:', error);

} finally {

setLoading(false);

}

}

}, [contract, account]);

// Event handlers for various actions

const handleRegisterAsPatient = async () => {

if (!contract) {

console.error("Contract not initialized");

alert("Please wait for the contract to initialize or try refreshing the page.");

return;

}

try {

setLoading(true);

const result = await contract.methods.registerAsPatient().send({ from: account });

console.log("Transaction result:", result);

alert('Successfully registered as a patient!');

fetchData();

} catch (error) {

console.error('Error registering as patient:', error);

alert(`Failed to register as a patient: ${error.message}`);

} finally {

setLoading(false);

}

};

const handleRegisterAsHealthcareExpert = async () => {

if (!contract) {

console.error("Contract not initialized");

alert("Please wait for the contract to initialize or try refreshing the page.");

return;

}

try {

setLoading(true);

const result = await contract.methods.registerAsHealthcareExpert("Doctor Name", "Specialization", 5)

.send({ from: account });

console.log("Transaction result:", result);

alert('Successfully registered as a healthcare expert!');

fetchData();

} catch (error) {

console.error('Error registering as healthcare expert:', error);

alert(`Failed to register as a healthcare expert: ${error.message}`);

} finally {

setLoading(false);

}

};

const handleAddHealthcareExpert = async (e) => {

e.preventDefault();

if (!contract) {

console.error("Contract not initialized");

alert("Please wait for the contract to initialize or try refreshing the page.");

return;

}

try {

setLoading(true);

const result = await contract.methods.addHealthcareExpert(expertAddress).send({ from: account });

console.log("Transaction result:", result);

alert('Successfully added healthcare expert!');

setExpertAddress('');

fetchData();

} catch (error) {

console.error('Error adding healthcare expert:', error);

alert(`Failed to add healthcare expert: ${error.message}`);

} finally {

setLoading(false);

}

};

const handleSendHealthData = async (e) => {

e.preventDefault();

if (!contract) {

console.error("Contract not initialized");

alert("Please wait for the contract to initialize or try refreshing the page.");

return;

}

try {

setLoading(true);

const result = await contract.methods.sendHealthData(healthData).send({ from: account });

console.log("Transaction result:", result);

alert('Successfully sent health data!');

setHealthData('');

fetchData();

} catch (error) {

console.error('Error sending health data:', error);

alert(`Failed to send health data: ${error.message}`);

} finally {

setLoading(false);

}

};

const handleSendMessageToPatient = async (e) => {

e.preventDefault();

if (!contract) {

console.error("Contract not initialized");

alert("Please wait for the contract to initialize or try refreshing the page.");

return;

}

try {

setLoading(true);

const result = await contract.methods.sendMessageToPatient(patientAddress, message)

.send({ from: account });

console.log("Transaction result:", result);

alert('Successfully sent message to patient!');

setMessage('');

setPatientAddress('');

} catch (error) {

console.error('Error sending message to patient:', error);

alert(`Failed to send message to patient: ${error.message}`);

} finally {

setLoading(false);

}

};

const handleAddFamilyMember = async (e) => {

e.preventDefault();

if (!contract) {

console.error("Contract not initialized");

alert("Please wait for the contract to initialize or try refreshing the page.");

return;

}

try {

setLoading(true);

const result = await contract.methods.addFamilyMember(familyMemberAddress).send({ from: account });

console.log("Transaction result:", result);

alert('Successfully added family member!');

setFamilyMemberAddress('');

fetchData();

} catch (error) {

console.error('Error adding family member:', error);

alert(`Failed to add family member: ${error.message}`);

} finally {

setLoading(false);

}

};

const handleRegisterAsResearchInstitute = async () => {

if (!contract) {

console.error("Contract not initialized");

alert("Please wait for the contract to initialize or try refreshing the page.");

return;

}

try {

setLoading(true);

const result = await contract.methods.registerAsResearchInstitute("Research Institute Name", "Research Area")

.send({ from: account });

console.log("Transaction result:", result);

alert('Successfully registered as a research institute!');

fetchData();

} catch (error) {

console.error('Error registering as research institute:', error);

alert(`Failed to register as a research institute: ${error.message}`);

} finally {

setLoading(false);

}

};

const handleSetConsentToRI = async (e) => {

e.preventDefault();

if (!contract) {

console.error("Contract not initialized");

alert("Please wait for the contract to initialize or try refreshing the page.");

return;

}

try {

setLoading(true);

const result = await contract.methods.setConsentToRI(consentToRI).send({ from: account });

console.log("Transaction result:", result);

alert('Consent to share data with Research Institute updated!');

setConsentToRI(false); // Reset consent input after successful transaction

fetchData();

} catch (error) {

console.error('Error setting consent to RI:', error);

alert(`Failed to set consent to RI: ${error.message}`);

} finally {

setLoading(false);

}

};

const handleRewardPatient = async (e) => {

e.preventDefault();

if (!contract) {

console.error("Contract not initialized");

alert("Please wait for the contract to initialize or try refreshing the page.");

return;

}

try {

setLoading(true);

const result = await contract.methods.rewardPatient(patientAddress).send({ from: account, value: web3.utils.toWei('1', 'ether') });

console.log("Transaction result:", result);

alert('Successfully rewarded patient with 1 ether!');

setRewardAmount(0);

} catch (error) {

console.error('Error rewarding patient:', error);

alert(`Failed to reward patient: ${error.message}`);

} finally {

setLoading(false);

}

};

return (

<div className="App">

<header className="App-header">

<h1>Health Data Management</h1>

<p>Account: {account}</p>

<div className="form-container">

<button onClick={handleRegisterAsPatient} disabled={loading}>

Register as Patient

</button>

<button onClick={handleRegisterAsHealthcareExpert} disabled={loading}>

Register as Healthcare Expert

</button>

<button onClick={handleRegisterAsResearchInstitute} disabled={loading}>

Register as Research Institute

</button>

</div>

</header>

<main>

<section>

<h2>Healthcare Experts</h2>

<ul>

{healthcareExperts.map((expert, index) => (

<li key={index}>{expert}</li>

))}

</ul>

<form onSubmit={handleAddHealthcareExpert}>

<input

type="text"

value={expertAddress}

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

placeholder="Healthcare Expert Address"

/>

<button type="submit" disabled={loading}>

Add Healthcare Expert

</button>

</form>

</section>

<section>

<h2>Patients</h2>

<ul>

{patients.map((patient, index) => (

<li key={index}>{patient}</li>

))}

</ul>

<form onSubmit={handleSendMessageToPatient}>

<input

type="text"

value={patientAddress}

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

placeholder="Patient Address"

/>

<input

type="text"

value={message}

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

placeholder="Message"

/>

<button type="submit" disabled={loading}>

Send Message to Patient

</button>

</form>

</section>

<section>

<h2>Notifications</h2>

<ul>

{notifications.map((notification, index) => (

<li key={index}>{notification}</li>

))}

</ul>

</section>

<section>

<h2>Family Members</h2>

<ul>

{familyMembers.map((member, index) => (

<li key={index}>{member}</li>

))}

</ul>

<form onSubmit={handleAddFamilyMember}>

<input

type="text"

value={familyMemberAddress}

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

placeholder="Family Member Address"

/>

<button type="submit" disabled={loading}>

Add Family Member

</button>

</form>

</section>

<section>

<h2>Research Institutes</h2>

<ul>

{researchInstitutes.map((ri, index) => (

<li key={index}>{ri}</li>

))}

</ul>

</section>

<section>

<form onSubmit={handleSendHealthData}>

<h2>Send Health Data</h2>

<textarea

value={healthData}

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

placeholder="Health Data"

/>

<button type="submit" disabled={loading}>

Send Health Data

</button>

</form>

</section>

<section>

<form onSubmit={handleSetConsentToRI}>

<h2>Set Consent to Research Institutes</h2>

<label>

<input

type="checkbox"

checked={consentToRI}

onChange={(e) => setConsentToRI(e.target.checked)}

/>

Consent to share data with Research Institutes

</label>

<button type="submit" disabled={loading}>

Set Consent

</button>

</form>

</section>

<section>

<form onSubmit={handleRewardPatient}>

<h2>Reward Patient</h2>

<input

type="text"

value={patientAddress}

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

placeholder="Patient Address"

/>

<input

type="number"

value={rewardAmount}

onChange={(e) => setRewardAmount(Number(e.target.value))}

placeholder="Reward Amount"

/>

<button type="submit" disabled={loading}>

Reward Patient

</button>

</form>

</section>

<section>

<form onSubmit={handleAddFamilyMember}>

<h2>Register as Family Member</h2>

<input

type="text"

value={familyMemberAddress}

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

placeholder="Family Member Address"

/>

<input

type="text"

value={familyMemberName}

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

placeholder="Family Member Name"

/>

<button type="submit" disabled={loading}>

Register as Family Member

</button>

</form>

</section>

</main>

{loading && <div className="loading">Processing...</div>}

</div>

);

};

export default App;