/ SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract HealthDataSharing {

struct HealthcareExpert {

string name;

string specialization;

uint yearsOfExperience;

bool isRegistered;

}

struct ResearchInstitute {

string name;

bool isRegistered;

}

struct FamilyMember {

string name;

bool isRegistered;

}

mapping(address => HealthcareExpert) public healthcareExperts;

mapping(address => bool) public registeredPatients;

mapping(address => ResearchInstitute) public researchInstitutes;

mapping(address => FamilyMember) public familyMembers;

mapping(address => address[]) public patientExpertsList;

mapping(address => mapping(address => bool)) public patientExperts;

mapping(address => address[]) public patientFamilyMembersList;

mapping(address => string[]) private patientNotifications;

mapping(address => uint8) public patientPrivacyScore;

mapping(address => uint) public lastHealthDataTime;

mapping(address => bool) public patientConsentToRI;

address[] private patientList;

address[] private expertList;

address[] private researchInstituteList;

address[] private familyMemberList;

event HealthcareExpertRegistered(address expert, string name, string specialization, uint yearsOfExperience);

event PatientRegistered(address patient);

event ResearchInstituteRegistered(address institute, string name);

event FamilyMemberRegistered(address familyMember, string name);

event HealthcareExpertAdded(address patient, address expert);

event HealthcareExpertRemoved(address patient, address expert);

event HealthDataSent(address patient, string data);

event MessageSent(address expert, address patient, string message);

event HealthDataReceived(address expert, string data);

event HealthDataRequest(address patient);

event PatientConsentToRI(address patient, bool consent);

event PatientRewarded(address patient, uint amount);

event HealthDataSentToRI(address expert, address patient, address researchInstitute, string data);

event FamilyMemberNotified(address indexed patient, address indexed familyMember, string action);

modifier onlyRegisteredExpert() {

require(healthcareExperts[msg.sender].isRegistered, "Caller is not a registered healthcare expert");

\_;

}

modifier onlyRegisteredPatient() {

require(registeredPatients[msg.sender], "Caller is not a registered patient");

\_;

}

modifier onlyRegisteredResearchInstitute() {

require(researchInstitutes[msg.sender].isRegistered, "Caller is not a registered research institute");

\_;

}

modifier onlyRegisteredFamilyMember() {

require(familyMembers[msg.sender].isRegistered, "Caller is not a registered family member");

\_;

}

function registerAsPatient() public {

require(!registeredPatients[msg.sender], "Patient is already registered");

registeredPatients[msg.sender] = true;

patientList.push(msg.sender);

emit PatientRegistered(msg.sender);

}

function registerAsHealthcareExpert(string memory name, string memory specialization, uint yearsOfExperience) public {

require(!healthcareExperts[msg.sender].isRegistered, "Expert is already registered");

healthcareExperts[msg.sender] = HealthcareExpert({

name: name,

specialization: specialization,

yearsOfExperience: yearsOfExperience,

isRegistered: true

});

expertList.push(msg.sender);

emit HealthcareExpertRegistered(msg.sender, name, specialization, yearsOfExperience);

}

function registerAsResearchInstitute(string memory name) public {

require(!researchInstitutes[msg.sender].isRegistered, "Institute is already registered");

researchInstitutes[msg.sender] = ResearchInstitute({

name: name,

isRegistered: true

});

researchInstituteList.push(msg.sender);

emit ResearchInstituteRegistered(msg.sender, name);

}

function registerAsFamilyMember(string memory name) public {

require(!familyMembers[msg.sender].isRegistered, "Family member is already registered");

familyMembers[msg.sender] = FamilyMember({

name: name,

isRegistered: true

});

familyMemberList.push(msg.sender);

emit FamilyMemberRegistered(msg.sender, name);

}

function addHealthcareExpert(address expert) public onlyRegisteredPatient {

require(healthcareExperts[expert].isRegistered, "Expert is not registered");

require(!patientExperts[msg.sender][expert], "Expert is already added");

patientExperts[msg.sender][expert] = true;

patientExpertsList[msg.sender].push(expert);

emit HealthcareExpertAdded(msg.sender, expert);

}

function removeHealthcareExpert(address expert) public onlyRegisteredPatient {

require(patientExperts[msg.sender][expert], "Expert is not assigned to this patient");

patientExperts[msg.sender][expert] = false;

address[] storage expertsList = patientExpertsList[msg.sender];

for (uint i = 0; i < expertsList.length; i++) {

if (expertsList[i] == expert) {

expertsList[i] = expertsList[expertsList.length - 1];

expertsList.pop();

break;

}

}

emit HealthcareExpertRemoved(msg.sender, expert);

}

function addFamilyMember(address familyMember) public onlyRegisteredPatient {

require(familyMembers[familyMember].isRegistered, "Family member is not registered");

patientFamilyMembersList[msg.sender].push(familyMember);

}

function sendHealthData(string memory data) public onlyRegisteredPatient {

lastHealthDataTime[msg.sender] = block.timestamp;

emit HealthDataSent(msg.sender, data);

address[] memory expertsList = patientExpertsList[msg.sender];

for (uint i = 0; i < expertsList.length; i++) {

if (patientExperts[msg.sender][expertsList[i]]) {

emit HealthDataReceived(expertsList[i], data);

}

}

address[] memory familyMembersList = patientFamilyMembersList[msg.sender];

for (uint i = 0; i < familyMembersList.length; i++) {

emit FamilyMemberNotified(msg.sender, familyMembersList[i], "Health data sent");

}

}

function sendMessageToPatient(address patient, string memory message) public onlyRegisteredExpert {

require(patientExperts[patient][msg.sender], "You are not authorized to send a message to this patient");

require(bytes(message).length > 0, "Message cannot be empty");

patientNotifications[patient].push(message);

emit MessageSent(msg.sender, patient, message);

address[] memory familyMembersList = patientFamilyMembersList[patient];

for (uint i = 0; i < familyMembersList.length; i++) {

emit FamilyMemberNotified(patient, familyMembersList[i], "Message received from healthcare expert");

}

}

function viewNotifications() public view onlyRegisteredPatient returns (string[] memory) {

string[] memory notifications = patientNotifications[msg.sender];

if (block.timestamp >= lastHealthDataTime[msg.sender] + 1 hours) {

string[] memory updatedNotifications = new string[](notifications.length + 1);

for (uint i = 0; i < notifications.length; i++) {

updatedNotifications[i] = notifications[i];

}

updatedNotifications[notifications.length] = "Reminder: Please send your health data.";

return updatedNotifications;

}

return notifications;

}

function checkHealthDataTime() public {

uint currentTime = block.timestamp;

for (uint i = 0; i < patientList.length; i++) {

address patient = patientList[i];

if (currentTime >= lastHealthDataTime[patient] + 1 hours) {

emit HealthDataRequest(patient);

}

}

}

// Function to set privacy score

function setPrivacyScore(uint8 score) public onlyRegisteredPatient {

require(score <= 5, "Privacy score must be between 0 and 5");

patientPrivacyScore[msg.sender] = score;

}

// Function to get privacy score

function getPrivacyScore(address patient) public view returns (uint8) {

return patientPrivacyScore[patient];

}

function getTotalHealthcareExperts() public view returns (uint) {

return expertList.length;

}

function getHealthcareExpertAddresses() public view returns (address[] memory) {

return expertList;

}

function getTotalPatients() public view returns (uint) {

return patientList.length;

}

function getPatientAddresses() public view returns (address[] memory) {

return patientList;

}

function getLastHealthDataTime(address patient) public view returns (uint) {

return lastHealthDataTime[patient];

}

function needsToSendHealthData(address patient) public view returns (bool) {

uint currentTime = block.timestamp;

return currentTime >= lastHealthDataTime[patient] + 1 hours;

}

function setConsentToRI(bool consent) public onlyRegisteredPatient {

patientConsentToRI[msg.sender] = consent;

emit PatientConsentToRI(msg.sender, consent);

}

function rewardPatient(address patient) public payable onlyRegisteredResearchInstitute {

require(patientConsentToRI[patient], "Patient has not given consent to share data with R.I.");

require(registeredPatients[patient], "Patient is not registered");

(bool success, ) = patient.call{value: msg.value}("");

require(success, "Transfer failed");

emit PatientRewarded(patient, msg.value);

}

// Modify the sendHealthDataToRI function to include family member notifications

function sendHealthDataToRI(address patient, address researchInstitute, string memory data) public onlyRegisteredExpert {

require(patientConsentToRI[patient], "Patient has not given consent to share data with R.I.");

require(patientExperts[patient][msg.sender], "You are not authorized to send data for this patient");

require(researchInstitutes[researchInstitute].isRegistered, "Research institute is not registered");

emit HealthDataSentToRI(msg.sender, patient, researchInstitute, data);

address[] memory familyMembersList = patientFamilyMembersList[patient];

for (uint i = 0; i < familyMembersList.length; i++) {

emit FamilyMemberNotified(patient, familyMembersList[i], "Health data sent to Research Institute");

}

}

}