// SPDX-License-Identifier: MIT

pragma solidity 0.8.26;

contract DecentralizedHealthcare {

    address public admin;

    constructor() {

        admin = msg.sender;

    }

    // Structs

    struct Patient {

        string name;

        bool isRegistered;

        mapping(address => bool) authorizedDoctors;

        mapping(address => bool) authorizedLabs;

        address[] doctorList;

        address[] labList;

    }

    struct Doctor {

        string name;

        string specialization;

        bool isRegistered;

    }

    struct Laboratory {

        string name;

        string labType;

        bool isRegistered;

    }

    struct Record {

        string ipfsHash;

        string description;

        uint256 timestamp;

        address addedBy;

    }

    struct Medication {

        string name;

        string dosage;

        string timing;

        string sideEffects;

        uint256 reminderInterval;

    }

    struct LabReport {

        string ipfsHash;

        string reportType;

        uint256 timestamp;

        address lab;

    }

    // Mappings

    mapping(address => Patient) private patients;

    mapping(address => Doctor) public doctors;

    mapping(address => Laboratory) public laboratories;

    mapping(address => Record[]) private patientRecords;

    mapping(address => Medication[]) private patientMedications;

    mapping(address => LabReport[]) private patientLabReports;

    // Events

    event PatientRegistered(address indexed patient, string name);

    event DoctorRegistered(address indexed doctor, string name, string specialization);

    event LaboratoryRegistered(address indexed lab, string name, string labType);

    event AccessGranted(address indexed patient, address indexed user, string role);

    event AccessRevoked(address indexed patient, address indexed user, string role);

    event RecordAdded(address indexed patient, address indexed doctor, string ipfsHash);

    event MedicationAdded(address indexed patient, string medicationName);

    event LabReportAdded(address indexed patient, address indexed lab, string ipfsHash, string reportType);

    // Modifiers

    modifier onlyAdmin() {

        require(msg.sender == admin, "Admin only");

        \_;

    }

    modifier onlyPatient() {

        require(patients[msg.sender].isRegistered, "Only patient allowed");

        \_;

    }

    modifier onlyDoctor() {

        require(doctors[msg.sender].isRegistered, "Only doctor allowed");

        \_;

    }

    modifier onlyLab() {

        require(laboratories[msg.sender].isRegistered, "Only lab allowed");

        \_;

    }

    modifier doctorHasAccess(address \_patient) {

        require(patients[\_patient].authorizedDoctors[msg.sender], "Doctor not authorized");

        \_;

    }

    modifier labHasAccess(address \_patient) {

        require(patients[\_patient].authorizedLabs[msg.sender], "Lab not authorized");

        \_;

    }

    // Registration

    function registerPatient(string memory \_name) external {

        require(!patients[msg.sender].isRegistered, "Already registered");

        patients[msg.sender].name = \_name;

        patients[msg.sender].isRegistered = true;

        emit PatientRegistered(msg.sender, \_name);

    }

    function registerDoctor(address \_doctor, string memory \_name, string memory \_specialization) external onlyAdmin {

        require(!doctors[\_doctor].isRegistered, "Doctor already registered");

        doctors[\_doctor] = Doctor(\_name, \_specialization, true);

        emit DoctorRegistered(\_doctor, \_name, \_specialization);

    }

    function registerLaboratory(address \_lab, string memory \_name, string memory \_type) external onlyAdmin {

        require(!laboratories[\_lab].isRegistered, "Lab already registered");

        laboratories[\_lab] = Laboratory(\_name, \_type, true);

        emit LaboratoryRegistered(\_lab, \_name, \_type);

    }

    // Access Control

    function grantDoctorAccess(address \_doctor) external onlyPatient {

        require(doctors[\_doctor].isRegistered, "Invalid doctor");

        patients[msg.sender].authorizedDoctors[\_doctor] = true;

        patients[msg.sender].doctorList.push(\_doctor);

        emit AccessGranted(msg.sender, \_doctor, "Doctor");

    }

    function revokeDoctorAccess(address \_doctor) external onlyPatient {

        require(patients[msg.sender].authorizedDoctors[\_doctor], "Not authorized");

        patients[msg.sender].authorizedDoctors[\_doctor] = false;

        emit AccessRevoked(msg.sender, \_doctor, "Doctor");

    }

    function grantLabAccess(address \_lab) external onlyPatient {

        require(laboratories[\_lab].isRegistered, "Invalid lab");

        patients[msg.sender].authorizedLabs[\_lab] = true;

        patients[msg.sender].labList.push(\_lab);

        emit AccessGranted(msg.sender, \_lab, "Lab");

    }

    function revokeLabAccess(address \_lab) external onlyPatient {

        require(patients[msg.sender].authorizedLabs[\_lab], "Lab not authorized");

        patients[msg.sender].authorizedLabs[\_lab] = false;

        emit AccessRevoked(msg.sender, \_lab, "Lab");

    }

    // Medical Records

    function addMedicalRecord(address \_patient, string memory \_ipfsHash, string memory \_description)

        external

        onlyDoctor

        doctorHasAccess(\_patient)

    {

        patientRecords[\_patient].push(Record(\_ipfsHash, \_description, block.timestamp, msg.sender));

        emit RecordAdded(\_patient, msg.sender, \_ipfsHash);

    }

    function getMedicalRecords() external view onlyPatient returns (Record[] memory) {

        return patientRecords[msg.sender];

    }

    function getPatientRecords(address \_patient) external view onlyDoctor doctorHasAccess(\_patient) returns (Record[] memory) {

        return patientRecords[\_patient];

    }

    // Medication

    function addMedication(address \_patient, string memory \_name, string memory \_dosage, string memory \_timing, string memory \_sideEffects, uint256 \_reminderInterval)

        external

        onlyDoctor

        doctorHasAccess(\_patient)

    {

        patientMedications[\_patient].push(Medication(\_name, \_dosage, \_timing, \_sideEffects, \_reminderInterval));

        emit MedicationAdded(\_patient, \_name);

    }

    function getMedications() external view onlyPatient returns (Medication[] memory) {

        return patientMedications[msg.sender];

    }

    // Lab Reports

    function addLabReport(address \_patient, string memory \_ipfsHash, string memory \_reportType)

        external

        onlyLab

        labHasAccess(\_patient)

    {

        patientLabReports[\_patient].push(LabReport(\_ipfsHash, \_reportType, block.timestamp, msg.sender));

        emit LabReportAdded(\_patient, msg.sender, \_ipfsHash, \_reportType);

    }

    function getMyLabReports() external view onlyPatient returns (LabReport[] memory) {

        return patientLabReports[msg.sender];

    }

    function getPatientLabReports(address \_patient) external view onlyDoctor doctorHasAccess(\_patient) returns (LabReport[] memory) {

        return patientLabReports[\_patient];

    }

    // Utility

    function getAuthorizedDoctors() external view onlyPatient returns (address[] memory) {

        return patients[msg.sender].doctorList;

    }

    function getAuthorizedLabs() external view onlyPatient returns (address[] memory) {

        return patients[msg.sender].labList;

    }

    function getPatientName(address \_addr) external view returns (string memory) {

        require(patients[\_addr].isRegistered, "Not registered");

        return patients[\_addr].name;

    }

    function getDoctorInfo(address \_addr) external view returns (string memory name, string memory specialization) {

        require(doctors[\_addr].isRegistered, "Not registered");

        Doctor memory doc = doctors[\_addr];

        return (doc.name, doc.specialization);

    }

    function getLabInfo(address \_addr) external view returns (string memory name, string memory labType) {

        require(laboratories[\_addr].isRegistered, "Not registered");

        Laboratory memory lab = laboratories[\_addr];

        return (lab.name, lab.labType);

    }

}