TELEMEDICINE

// SPDX-License-Identifier: MIT pragma solidity ^0.8.0;

contract Telemedicine { struct Record { string ipfsHash; address owner; }

```
struct Appointment {
    address patient;
    address doctor;
    uint256 timestamp;
    string description;
    bool confirmed;
    bool canceled;
    uint256 fee;
}

struct TelemedicineSession {
    uint256 appointmentId;
    string videoCallLink;
    bool active;
}

struct Prescription {
    uint256 appointmentId;
    string ipfsHash;
    address doctor;
}

mapping(address => Record[]) private records;
mapping(address => mapping(address => bool)) private permissions;
mapping(uint256 => Appointment) public appointments;
mapping(uint256 => TelemedicineSession) public sessions;
mapping(uint256 => Prescription) public prescriptions;

uint256 public appointmentCount;
uint256 public sessionCount;
uint256 public prescriptionCount;

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

```
event PermissionGranted(address indexed patient, address indexed
doctor);
event PermissionRevoked(address indexed patient, address indexed
doctor);
event AppointmentScheduled(uint256 indexed appointmentId, address
indexed patient, address indexed doctor, uint256 timestamp, string
description, uint256 fee);
event AppointmentConfirmed(uint256 indexed appointmentId);
event AppointmentCanceled(uint256 indexed appointmentId);
event TelemedicineSessionStarted(uint256 indexed sessionId, uint256
indexed appointmentId, string videoCallLink);
event PrescriptionAdded(uint256 indexed prescriptionId, uint256
indexed appointmentId, string ipfsHash);
event PaymentSent(address indexed from, address indexed to, uint256
amount);

function addRecord(string memory _ipfsHash) public {
    records[msg.sender].push(Record(_ipfsHash, msg.sender));
    emit RecordAdded(msg.sender, _ipfsHash);
}

function grantAccess(address _doctor) public {
    permissions[msg.sender][_doctor] = true;
    emit PermissionGranted(msg.sender, _doctor);
}

function revokeAccess(address _doctor) public {
    permissions[msg.sender][_doctor] = false;
    emit PermissionRevoked(msg.sender, _doctor);
}

function getRecords(address _patient) public view returns (Record[]
memory) {
    require(permissions[_patient][msg.sender] || msg.sender ==
_patient, "Access Denied");
    return records[_patient];
}

function scheduleAppointment(address _doctor, uint256 _timestamp,
```

```
string memory _description) public payable {
    require(msg.value > 0, "Payment required");
    appointments[appointmentCount] = Appointment(msg.sender, _doctor,
_timestamp, _description, false, false, msg.value);
    emit AppointmentScheduled(appointmentCount, msg.sender, _doctor,
_timestamp, _description, msg.value);
    appointmentCount++;
}

function confirmAppointment(uint256 _appointmentId) public {
    require(appointments[_appointmentId].doctor == msg.sender, "Only
doctor can confirm");
    require(!appointments[_appointmentId].canceled, "Appointment
canceled");
    appointments[_appointmentId].confirmed = true;
    emit AppointmentConfirmed(_appointmentId);
}

function cancelAppointment(uint256 _appointmentId) public {
    require(appointments[_appointmentId].patient == msg.sender ||
appointments[_appointmentId].doctor == msg.sender, "Unauthorized");
    require(!appointments[_appointmentId].canceled, "Already
canceled");
    appointments[_appointmentId].canceled = true;

payable(appointments[_appointmentId].patient).transfer(appointments[_a
ppointmentId].fee);
    emit AppointmentCanceled(_appointmentId);
}

function startTelemedicineSession(uint256 _appointmentId, string
memory _videoCallLink) public {
    require(appointments[_appointmentId].confirmed, "Appointment not
confirmed");
    require(!appointments[_appointmentId].canceled, "Appointment
canceled");
    require(appointments[_appointmentId].doctor == msg.sender ||
appointments[_appointmentId].patient == msg.sender, "Unauthorized");
```

```
    sessions[sessionCount] = TelemedicineSession(_appointmentId,
_videoCallLink, true);
    emit TelemedicineSessionStarted(sessionCount, _appointmentId,
_videoCallLink);
    sessionCount++;
}

function addPrescription(uint256 _appointmentId, string memory
_ipfsHash) public {
    require(appointments[_appointmentId].doctor == msg.sender, "Only
doctor can add prescription");
    require(appointments[_appointmentId].confirmed, "Appointment not
confirmed");
    require(!appointments[_appointmentId].canceled, "Appointment
canceled");

    prescriptions[prescriptionCount] = Prescription(_appointmentId,
_ipfsHash, msg.sender);
    emit PrescriptionAdded(prescriptionCount, _appointmentId,
_ipfsHash);
    prescriptionCount++;
}

function sendPayment(uint256 _appointmentId) public {
    require(appointments[_appointmentId].patient == msg.sender, "Only
patient can pay");
    require(appointments[_appointmentId].confirmed, "Appointment not
confirmed");
    require(!appointments[_appointmentId].canceled, "Appointment
canceled");
    require(address(this).balance >= appointments[_appointmentId].fee,
"Insufficient balance");


payable(appointments[_appointmentId].doctor).transfer(appointments[_ap
pointmentId].fee);
    emit PaymentSent(msg.sender, appointments[_appointmentId].doctor,
appointments[_appointmentId].fee);
```

```
}

}
```