**✅ What Was Implemented**

**1. 📤 Record Upload Functionality**

* Each patient can upload multiple records.
* Each record includes:
  + description → e.g., “Blood Report”
  + ipfsHash → Pinata CID (content stored on IPFS)
  + uploadedBy → The address of the uploader

**Smart Contract Function:**

solidity

CopyEdit

function addRecord(string memory \_description, string memory \_ipfsHash) public {

Record memory newRecord = Record(\_description, \_ipfsHash, msg.sender);

patientRecords[msg.sender].push(newRecord);

}

**2. 🔒 Access Control System**

* Patients grant doctors access manually via grantAccess(address doctor)
* Only allowed addresses can view patient records.

**Smart Contract Functions:**

solidity

CopyEdit

function grantAccess(address doctor) public {

access[msg.sender][doctor] = true;

}

function viewRecord(address patient) public view returns (Record[] memory) {

require(access[patient][msg.sender], "No Access!");

return patientRecords[patient];

}

**🧪 Testing and Observations**

**✅ Used Two Remix Accounts:**

* **Account 1 (Patient):**
  + Uploaded a record
  + Granted access to Account 2
* **Account 2 (Doctor):**
  + Could only view the record after access was granted
  + Got "No Access!" error before access

**🗂️ IPFS & Pinata**

* Used **Pinata** to manually upload a dummy report (blood\_report.txt)
* Received a CID:  
  Example → bafkreihdwdcefgh4dqkjv67uzcmw7ojee6xedzdetojuzjevtenxquvyku
* Used this CID in the addRecord() function call

**🔍 Learnings**

* Solidity mapping inside mapping for access management
* IPFS stores files in a decentralized way; only the CID is stored on-chain
* require() helps enforce access security in smart contracts
* Remix VM environment can simulate multi-user behavior via multiple accounts
* No receive() or fallback() is needed right now since no ether is being sent