Set B

Practical 1.

Develop a secure and decentralized land registration smart contract that allows:
The government authority (land registrar) to register land properties.
Owners to transfer land ownership to other individuals securely.

LandRegistrar: The government authority that registers land and approves transfers.
Owner (User): Person who owns a piece of land and can transfer it.
Buyer/New Owner: A valid Ethereum address to whom land is transferred.
Functional requirements:
Only the LandRegistrar (contract deployer) can register a new land property.
Land properties must include: Land ID, Location (string), Current Owner, Area (in square meters),
Owner of a land can initiate a transfer to another address.
LandRegistrar must approve the transfer to finalize ownership change.
System must prevent unauthorized land registration or transfer.
Submit:
Source code (.sol file)
Deployment screenshots (e.g., Remix IDE)
Sample test transactions (Land registration and ownership transfer)

Code:
LandRegistry.sol

```
pragma solidity ^0.8.0;
contract LandRegistry {
    address public landRegistrar;

    struct Land {
        uint256 id;
        string location;
        address currentOwner;
        uint256 area;
        bool isRegistered;
    }

    struct TransferRequest {
        uint256 landId;
        address from;
        address to;
```

```
        bool isPending;
    }

    mapping(uint256 => Land) public lands;

    mapping(uint256 => TransferRequest) public transferRequests;

    event LandRegistered(uint256 indexed landId, string location, address
indexed owner, uint256 area);
    event TransferRequested(uint256 indexed landId, address indexed from,
address indexed to);
    event TransferApproved(uint256 indexed landId, address indexed from,
address indexed to);

    modifier onlyLandRegistrar() {
        require(msg.sender == landRegistrar, "Only land registrar can
perform this action");
        _;
    }

    modifier onlyLandOwner(uint256 _landId) {
        require(lands[_landId].currentOwner == msg.sender, "Only land owner
can perform this action");
        require(lands[_landId].isRegistered, "Land is not registered");
        _;
    }
        constructor() {
        landRegistrar = msg.sender;
    }


    function registerLand(
        uint256 _landId,
        string memory _location,
        address _owner,
        uint256 _area
    ) public onlyLandRegistrar {
        require(!lands[_landId].isRegistered, "Land already registered");
        require(_owner != address(0), "Invalid owner address");
        require(_area > 0, "Area must be greater than zero");

        lands[_landId] = Land({
            id: _landId,
```

```
            location: _location,
            currentOwner: _owner,
            area: _area,
            isRegistered: true
        });

        emit LandRegistered(_landId, _location, _owner, _area);
    }


    function initiateTransfer(uint256 _landId, address _newOwner) public
onlyLandOwner(_landId) {
        require(_newOwner != address(0), "Invalid new owner address");
        require(_newOwner != lands[_landId].currentOwner, "New owner cannot
be the current owner");
        require(!transferRequests[_landId].isPending, "Transfer already
pending for this land");

        transferRequests[_landId] = TransferRequest({
            landId: _landId,
            from: msg.sender,
            to: _newOwner,
            isPending: true
        });

        emit TransferRequested(_landId, msg.sender, _newOwner);
    }


    function approveTransfer(uint256 _landId) public onlyLandRegistrar {
        require(lands[_landId].isRegistered, "Land is not registered");
        require(transferRequests[_landId].isPending, "No pending transfer
for this land");

        TransferRequest memory request = transferRequests[_landId];


        lands[_landId].currentOwner = request.to;


        transferRequests[_landId].isPending = false;

        emit TransferApproved(_landId, request.from, request.to);
```

```
    }


    function cancelTransfer(uint256 _landId) public {
        require(transferRequests[_landId].isPending, "No pending transfer
for this land");
        require(
            msg.sender == transferRequests[_landId].from || msg.sender ==
landRegistrar,
            "Only land owner or registrar can cancel transfer"
        );

        transferRequests[_landId].isPending = false;
    }

        function getLandDetails(uint256 _landId) public view returns (
        uint256 id,
        string memory location,
        address currentOwner,
        uint256 area,
        bool isRegistered
    ) {
        Land memory land = lands[_landId];
        return (
            land.id,
            land.location,
            land.currentOwner,
            land.area,
            land.isRegistered
        );
    }

    function getTransferRequest(uint256 _landId) public view returns (
        bool isPending,
        address from,
        address to
    ) {
        TransferRequest memory request = transferRequests[_landId];
        return (request.isPending, request.from, request.to);
    }
}
```

land_registry_test.js

```
const LandRegistry = artifacts.require("LandRegistry");

contract("LandRegistry", (accounts) => {
 const landRegistrar = accounts[0];
 const owner1 = accounts[1];
 const owner2 = accounts[2];
  let landRegistryInstance;
 const landId = 1;
 const location = "123 Main St, City";
 const area = 1000; // square meters

 beforeEach(async () => {
   landRegistryInstance = await LandRegistry.new({ from: landRegistrar });
 });

 it("should set the correct landRegistrar", async () => {
   const registrarAddress = await landRegistryInstance.landRegistrar();
   assert.equal(registrarAddress, landRegistrar, "Land registrar was not
correctly set");
 });

 it("should register land correctly", async () => {
   await landRegistryInstance.registerLand(landId, location, owner1, area, {
from: landRegistrar });

   const landDetails = await landRegistryInstance.getLandDetails(landId);
   assert.equal(landDetails.id, landId, "Land ID was not set correctly");
   assert.equal(landDetails.location, location, "Location was not set
correctly");
   assert.equal(landDetails.currentOwner, owner1, "Owner was not set
correctly");
   assert.equal(landDetails.area, area, "Area was not set correctly");
   assert.equal(landDetails.isRegistered, true, "Land was not registered");
 });

 it("should not allow non-registrar to register land", async () => {
   try {
     await landRegistryInstance.registerLand(landId, location, owner1, area, {
from: owner1 });
     assert.fail("The transaction should have thrown an error");
   } catch (err) {
     assert.include(err.message, "Only land registrar can perform this action",
```

```
"The error message is not correct");
    }
 });

 it("should allow owner to initiate transfer", async () => {
    await landRegistryInstance.registerLand(landId, location, owner1, area, {
from: landRegistrar });
    await landRegistryInstance.initiateTransfer(landId, owner2, { from: owner1
});

    const transferRequest = await
landRegistryInstance.getTransferRequest(landId);
    assert.equal(transferRequest.isPending, true, "Transfer request should be
pending");
    assert.equal(transferRequest.from, owner1, "From address is not correct");
    assert.equal(transferRequest.to, owner2, "To address is not correct");
 });

 it("should allow registrar to approve transfer", async () => {
    await landRegistryInstance.registerLand(landId, location, owner1, area, {
from: landRegistrar });
    await landRegistryInstance.initiateTransfer(landId, owner2, { from: owner1
});
    await landRegistryInstance.approveTransfer(landId, { from: landRegistrar });

    const landDetails = await landRegistryInstance.getLandDetails(landId);
    assert.equal(landDetails.currentOwner, owner2, "Ownership was not
transferred");

    const transferRequest = await
landRegistryInstance.getTransferRequest(landId);
    assert.equal(transferRequest.isPending, false, "Transfer request should not
be pending anymore");
 });
});
```

Output:


Compiling your contracts...
> Everything is up to date, there is nothing to compile.
Contract: LandRegistry
✔ should set the correct landRegistrar
✔ should register land correctly (207ms)
✔ should not allow non-registrar to register land (256ms)
✔ should allow owner to initiate transfer (218ms)
✔ should allow registrar to approve transfer (337ms)
Contract: LandRegistry
Land Registration
✔ should allow only the registrar to register land (227ms)
✔ should prevent registering the same land twice (132ms)
✔ should prevent invalid land registration (70ms)
Land Transfer
✔ should allow owner to initiate transfer (104ms)
✔ should prevent non-owner from initiating transfer
✔ should prevent transfer to invalid address
✔ should prevent transfer to current owner
✔ should prevent multiple pending transfers for same land (240ms)
Transfer Approval
✔ should allow registrar to approve transfer (144ms)
✔ should prevent unauthorized approval (39ms)
✔ should validate land registration for approval
✔ should prevent approval without pending transfer (133ms)
Transfer Cancellation
✔ should allow owner to cancel transfer (97ms)
✔ should allow registrar to cancel transfer (97ms)
✔ should prevent unauthorized cancellation
✔ should prevent cancellation when no transfer is pending (112ms)
Complete Land Transfer Workflow
✔ should handle a complete land registration and transfer cycle (575ms)
22 passing (8s)
shubham@shubham:~/Desktop/blockchain$

```
Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


Starting migrations...
======================
> Network name:    'development'
> Network id:      1746508269652
> Block gas limit: 30000000 (0x1c9c380)


1_deploy_land_registry.js
=========================

   Replacing 'LandRegistry'
   ------------------------
   > transaction hash:    0xcb30b0cc8d2c7ff72aba5eab7ebb8bd52262f29f879bdf7615655c6372a29e0b
   > Blocks: 0            Seconds: 0
   > contract address:    0xC7EF2A47ba33bD70d3e88BC6E4Ad6350c7691d5b
   > block number:        2
   > block timestamp:     1746508559
   > account:             0xda68e2BE93429Dc593969B42f7Fc8F1EAab4Ad98
   > balance:             999.99375799729131738
   > gas used:            939004 (0xe53fc)
   > gas price:           3.272471905 gwei
   > value sent:          0 ETH
   > total cost:          0.00307286420868262 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:     0.00307286420868262 ETH

Summary
=======
> Total deployments:   1
> Final cost:          0.00307286420868262 ETH

shubham@shubham:~/Desktop/blockchain$
```

```
shubham@shubham:~/Desktop/blockchain$ npm i
(################) ⁖ reify:supports-preserve-symlinks-flag: http fetch GET 200 https://registry.npmjs.org/supports-preserve-symlinks-flag/-/supports-preserve-s

shubham@shubham:~/Desktop/blockchain$ ls
contracts  LandRegistry.sol  migrations  node_modules  package.json  package-lock.json  README.md  test  truffle-config.js
shubham@shubham:~/Desktop/blockchain$  npx truffle compile

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.
shubham@shubham:~/Desktop/blockchain$ npx ganache --port 7545
This version of µWS is not compatible with your Node.js build:

Error: Cannot find module '../binaries/uws_linux_x64_109.node'
Require stack:
- /home/shubham/Desktop/blockchain/node_modules/ganache/node_modules/@trufflesuite/uws-js-unofficial/src/uws.js
- /home/shubham/Desktop/blockchain/node_modules/ganache/dist/node/cli.js
Falling back to a NodeJS implementation; performance may be degraded.


ganache v7.9.1 (@ganache/cli: 0.10.1, @ganache/core: 0.10.1)
Starting RPC server

Available Accounts
==================
(0) 0xda68e2BE93429Dc593969B42f7Fc8F1EAab4Ad98 (1000 ETH)
(1) 0xaAc1f1ae2e13c5a3C3C1E9723542381D7C14a299 (1000 ETH)
(2) 0xef75c6Cd2813Ef8217302B75A312b3f9AcBB3289 (1000 ETH)
(3) 0x15B8326014A9c88655F322EbB80A1Cfb4fa6c02C (1000 ETH)
(4) 0x179B0878F889ad4d9Cd80676C99C299ad938e7ED (1000 ETH)
(5) 0x047041F0e34F41E7a5d0f5995068c9CB24b0D850 (1000 ETH)
(6) 0x53Ea22Ba294de337966d2C88370bA5875DD57F6E (1000 ETH)
(7) 0x695C06fc4825c5D28584A0fd0Ebf78EDE3bafe37 (1000 ETH)
(8) 0xC7967e0D48C06E33BA884cE6dc7E60e15Fcc9e1B (1000 ETH)
(9) 0xe07BE18AC09D0F29cc1F0AD8De2355Af1485cce9 (1000 ETH)

Private Keys
==================
(0) 0x0658f5b7651193c239912d6aaf0d2d17f1788b34f1cc22673a60e2e39a719609
(1) 0x48dd1cb5d077deaab34f5b157956306d61048b126b40997869b521e36532843d
(2) 0xf138ed8e5c0341eb93da92c3e9cf4268d4ff05699bdc01a9644312bc11c124f0
(3) 0xe00a90104be9cb7b6e152151cd636428cb29c06a17fbc3a6ad9eaf5a05d7eeca
```

```
HD Wallet
==================
Mnemonic:       tunnel custom hockey lumber find similar domain predict wine witness path pause
Base HD Path:  m/44'/60'/0'/0/{account_index}

Default Gas Price
==================
2000000000

BlockGas Limit
==================
30000000

Call Gas Limit
==================
50000000

Chain
==================
Hardfork: shanghai
Id:       1337

RPC Listening on 127.0.0.1:7545
eth_blockNumber
net_version
eth_accounts
eth_getBlockByNumber
eth_accounts
net_version
eth_getBlockByNumber
eth_getBlockByNumber
net_version
eth_getBlockByNumber
eth_estimateGas
net_version
eth_blockNumber
eth_getBlockByNumber
eth_estimateGas
eth_getBlockByNumber
eth_gasPrice
eth_sendTransaction
```

```
  Transaction: 0xb3b986745f8c0cc4d47e6e71dbabfc8a2093ea1ca77bffcd966c3bc470c1bfc1
  Contract created: 0x5579bbe07348a88fd77636a9047e1cf2288d109e
  Gas usage: 939004
  Block number: 1
  Block time: Tue May 06 2025 10:41:49 GMT+0530 (India Standard Time)

eth_getTransactionReceipt
eth_getCode
eth_getTransactionByHash
eth_getBlockByNumber
eth_getBalance
eth_blockNumber
net_version
eth_accounts
eth_getBlockByNumber
eth_accounts
eth_getBlockByNumber
eth_getBlockByNumber
eth_getBlockByNumber
eth_estimateGas
(node:28269) MaxListenersExceededWarning: Possible EventEmitter memory leak detected. 11 close listeners added to [Server]. Use emitter.setMaxListeners() to increase limit
(Use `node --trace-warnings ...` to show where the warning was created)
eth_getBlockByNumber
eth_blockNumber
eth_estimateGas
eth_getBlockByNumber
eth_gasPrice
eth_sendTransaction

  Transaction: 0xcb30b0cc8d2c7ff72aba5eab7ebb8bd52262f29f879bdf7615655c6372a29e0b
  Contract created: 0xc7ef2a47ba33bd70d3e88bc6e4ad6350c7691d5b
  Gas usage: 939004
  Block number: 2
  Block time: Tue May 06 2025 10:45:59 GMT+0530 (India Standard Time)

eth_getTransactionReceipt
eth_getCode
eth_getTransactionByHash
eth_getBlockByNumber
eth_getBalance
```

Practical 2

Develop a decentralized smart contract system to manage patient medical records on the Ethereum blockchain. The contract should allow doctors to create records, patients to access them, and administrators to manage permissions.
Admin (Hospital or Health Authority): Deployer of the contract, manages doctor and patient registration.
Doctor: Authorized to create and update medical records.
Patients: Can view their own medical records only.
Functional requirements:
The Admin can register new doctors and patients.
Only authorized doctors can add or update a patient's medical record.
Only registered patients can view their own records.
A medical record should contain: Patient ID (address), Doctor ID (address), Diagnosis (string),
Prescription (string), Timestamp
Ensure access control so that no unauthorized access or modification occurs.

Code:
MedicalRecords.sol

```
pragma solidity ^0.8.0;
contract MedicalRecords {

    address public admin;


    mapping(address => bool) public registeredDoctors;

      mapping(address => bool) public registeredPatients;


    struct MedicalRecord {
        address patientId;
        address doctorId;
        string diagnosis;
        string prescription;
```

```
        uint256 timestamp;
    }

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


    event DoctorRegistered(address indexed doctorId);
    event PatientRegistered(address indexed patientId);
    event MedicalRecordAdded(address indexed patientId, address
indexed doctorId, uint256 timestamp);


    modifier onlyAdmin() {
        require(msg.sender == admin, "Only admin can perform this
action");
        _;
    }

    modifier onlyDoctor() {
        require(registeredDoctors[msg.sender], "Only registered
doctors can perform this action");
        _;
    }

    modifier onlyPatient(address patientId) {
        require(msg.sender == patientId, "You can only access your
own medical records");
        _;
    }

    modifier onlyRegisteredPatient() {
        require(registeredPatients[msg.sender], "Only registered
patients can perform this action");
        _;
    }

       constructor() {
        admin = msg.sender;
    }
```

```
    function registerDoctor(address doctorId) external onlyAdmin {
        require(doctorId != address(0), "Invalid doctor address");
        require(!registeredDoctors[doctorId], "Doctor already
registered");

        registeredDoctors[doctorId] = true;
        emit DoctorRegistered(doctorId);
    }

    function registerPatient(address patientId) external onlyAdmin {
        require(patientId != address(0), "Invalid patient address");
        require(!registeredPatients[patientId], "Patient already
registered");

        registeredPatients[patientId] = true;
        emit PatientRegistered(patientId);
    }

       function addMedicalRecord(address patientId, string memory
diagnosis, string memory prescription)
        external
        onlyDoctor
    {
        require(registeredPatients[patientId], "Patient is not
registered");

        MedicalRecord memory newRecord = MedicalRecord({
            patientId: patientId,
            doctorId: msg.sender,
            diagnosis: diagnosis,
            prescription: prescription,
            timestamp: block.timestamp
        });

        patientRecords[patientId].push(newRecord);
        emit MedicalRecordAdded(patientId, msg.sender,
block.timestamp);
```

```
    }


    function getMedicalRecordCount(address patientId)
        external
        view
        returns (uint256)
    {
        require(msg.sender == admin || msg.sender == patientId ||
registeredDoctors[msg.sender],
                "Unauthorized access");

        return patientRecords[patientId].length;
    }

      function getMedicalRecord(address patientId, uint256 index)
        external
        view
        returns (
            address,
            address,
            string memory,
            string memory,
            uint256
        )
    {
        require(msg.sender == admin || msg.sender == patientId ||
registeredDoctors[msg.sender],
                "Unauthorized access");
        require(index < patientRecords[patientId].length, "Record
index out of bounds");

        MedicalRecord memory record =
patientRecords[patientId][index];

        return (
            record.patientId,
            record.doctorId,
            record.diagnosis,
```

```
            record.prescription,
            record.timestamp
        );
    }

    function getMyMedicalRecords()
        external
        view
        onlyRegisteredPatient
        returns (MedicalRecord[] memory)
    {
        return patientRecords[msg.sender];
    }
       function updateMedicalRecord(
        address patientId,
        uint256 index,
        string memory diagnosis,
        string memory prescription
    )
        external
        onlyDoctor
    {
        require(registeredPatients[patientId], "Patient is not
registered");
        require(index < patientRecords[patientId].length, "Record
index out of bounds");

        MedicalRecord storage record =
patientRecords[patientId][index];
                require(record.doctorId == msg.sender, "Only the
doctor who created this record can update it");

        record.diagnosis = diagnosis;
        record.prescription = prescription;
        record.timestamp = block.timestamp;
    }

    function isDoctor(address doctorId) external view returns (bool)
{
```

```
        return registeredDoctors[doctorId];
    }


    function isPatient(address patientId) external view returns
(bool) {
        return registeredPatients[patientId];
    }
}
```

deploy.js

```
const hre = require("hardhat");

async function main() {
 console.log("Deploying MedicalRecords contract...");
 const MedicalRecords = await hre.ethers.getContractFactory("MedicalRecords");
 const medicalRecords = await MedicalRecords.deploy();

 await medicalRecords.waitForDeployment();
 const medicalRecordsAddress = await medicalRecords.getAddress();
 console.log(`MedicalRecords deployed to: ${medicalRecordsAddress}`);

 const [admin, doctor1, doctor2, patient1, patient2] = await
hre.ethers.getSigners();
 console.log(`Admin address: ${admin.address}`);

 console.log("\nRegistering a doctor...");
 const tx1 = await medicalRecords.registerDoctor(doctor1.address);
 await tx1.wait();
 console.log(`Doctor registered: ${doctor1.address}`);

 const isDoctor = await medicalRecords.isDoctor(doctor1.address);
 console.log(`Is doctor registered? ${isDoctor}`);

 console.log("\nRegistering a patient...");
 const tx2 = await medicalRecords.registerPatient(patient1.address);
 await tx2.wait();
 console.log(`Patient registered: ${patient1.address}`);

 const isPatient = await medicalRecords.isPatient(patient1.address);
```

```
 console.log(`Is patient registered? ${isPatient}`);

 console.log("\nDoctor adding a medical record...");
 const doctorContract = medicalRecords.connect(doctor1);
 const tx3 = await doctorContract.addMedicalRecord(
   patient1.address,
   "Seasonal Flu",
   "Rest, fluids, and antipyretics"
 );
 await tx3.wait();
 console.log("Medical record added");

 const recordCount = await
medicalRecords.getMedicalRecordCount(patient1.address);
 console.log(`Medical record count for patient: ${recordCount}`);

 const record = await medicalRecords.getMedicalRecord(patient1.address, 0);
 console.log("\nMedical Record Details:");
 console.log(`Patient ID: ${record[0]}`);
 console.log(`Doctor ID: ${record[1]}`);
 console.log(`Diagnosis: ${record[2]}`);
 console.log(`Prescription: ${record[3]}`);
 console.log(`Timestamp: ${record[4]}`);

 console.log("\nDoctor updating the medical record...");
 const tx4 = await doctorContract.updateMedicalRecord(
   patient1.address,
   0,
   "Influenza Type A",
   "Rest, increased fluid intake, and prescribed medication"
 );
 await tx4.wait();
 console.log("Medical record updated");
  // Get updated record
 const updatedRecord = await medicalRecords.getMedicalRecord(patient1.address,
0);
 console.log("\nUpdated Medical Record Details:");
 console.log(`Patient ID: ${updatedRecord[0]}`);
 console.log(`Doctor ID: ${updatedRecord[1]}`);
 console.log(`Diagnosis: ${updatedRecord[2]}`);
 console.log(`Prescription: ${updatedRecord[3]}`);
 console.log(`Timestamp: ${updatedRecord[4]}`);

 console.log("\nPatient viewing their records...");
 const patientContract = medicalRecords.connect(patient1);
```

```
 try {
   const patientRecords = await patientContract.getMyMedicalRecords();
   console.log("Patient successfully retrieved their records");
 } catch (error) {
   console.error("Error when patient tries to view records:", error);
 }

 console.log("\nTesting unauthorized access...");

 const unauthorizedDoctor = medicalRecords.connect(doctor2);
 try {
   await unauthorizedDoctor.addMedicalRecord(
     patient1.address,
     "Test diagnosis",
     "Test prescription"
   );
   console.log("ERROR: Unauthorized doctor was able to add a record");
 } catch (error) {
   console.log("✓ Unauthorized doctor correctly prevented from adding
records");
 }

 const unauthorizedPatient = medicalRecords.connect(patient2);
 try {
   await unauthorizedPatient.getMyMedicalRecords();
   console.log("ERROR: Unauthorized patient was able to view records");
 } catch (error) {
   console.log("✓ Unauthorized patient correctly prevented from viewing
records");
 }
  console.log("\nAll tests completed successfully!");
}

main()
 .then(() => process.exit(0))
 .catch((error) => {
   console.error(error);
   process.exit(1);
 });
```

Output:

```
shubham@shubham:~/Desktop/bk$ pkill -f "npx hardhat init"
shubham@shubham:~/Desktop/bk$ npx hardhat run scripts/deploy.js --network hardhat
Deploying MedicalRecords contract...
MedicalRecords deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
Admin address: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266

Registering a doctor...
Doctor registered: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
Is doctor registered? true

Registering a patient...
Patient registered: 0x90F79bf6EB2c4f870365E785982E1f101E93b906
Is patient registered? true

Doctor adding a medical record...
Medical record added
Medical record count for patient: 1

Medical Record Details:
Patient ID: 0x90F79bf6EB2c4f870365E785982E1f101E93b906
Doctor ID: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
Diagnosis: Seasonal Flu
Prescription: Rest, fluids, and antipyretics
Timestamp: 1746512385

Doctor updating the medical record...
Medical record updated

Updated Medical Record Details:
Patient ID: 0x90F79bf6EB2c4f870365E785982E1f101E93b906
Doctor ID: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
Diagnosis: Influenza Type A
Prescription: Rest, increased fluid intake, and prescribed medication
Timestamp: 1746512386

Patient viewing their records...
Patient successfully retrieved their records

Testing unauthorized access...
✓ Unauthorized doctor correctly prevented from adding records
✓ Unauthorized patient correctly prevented from viewing records

All tests completed successfully!
shubham@shubham:~/Desktop/bk$
```

```
Testing unauthorized access...
✓ Unauthorized doctor correctly prevented from adding records
✓ Unauthorized patient correctly prevented from viewing records

All tests completed successfully!
shubham@shubham:~/Desktop/bk$ npx hardhat run scripts/deploy.js --network hardhat
Deploying MedicalRecords contract...
MedicalRecords deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
Admin address: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266

Registering a doctor...
Doctor registered: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
Is doctor registered? true

Registering a patient...
Patient registered: 0x90F79bf6EB2c4f870365E785982E1f101E93b906
Is patient registered? true

Doctor adding a medical record...
Medical record added

Doctor adding a second medical record...
Second medical record added
Medical record count for patient: 2

Medical Record Details:
Patient ID: 0x90F79bf6EB2c4f870365E785982E1f101E93b906
Doctor ID: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
Diagnosis: Seasonal Flu
Prescription: Rest, fluids, and antipyretics
Timestamp: 1746512547

Doctor updating the medical record...
Medical record updated

Updated Medical Record Details:
Patient ID: 0x90F79bf6EB2c4f870365E785982E1f101E93b906
Doctor ID: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
Diagnosis: Influenza Type A
Prescription: Rest, increased fluid intake, and prescribed medication
Timestamp: 1746512549

Patient viewing their records...
Patient successfully retrieved their records

Testing unauthorized access...
✓ Unauthorized doctor correctly prevented from adding records
✓ Unauthorized patient correctly prevented from viewing records

All tests completed successfully!
shubham@shubham:~/Desktop/bk$
```

```
Is patient registered? true

Doctor adding a medical record...
Medical record added

Doctor adding a second medical record...
Second medical record added
Medical record count for patient: 2

Medical Record Details:
Patient ID: 0x90F79bf6EB2c4f870365E785982E1f101E93b906
Doctor ID: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
Diagnosis: Seasonal Flu
Prescription: Rest, fluids, and antipyretics
Timestamp: 1746512646

Doctor updating the medical record...
Medical record updated

Updated Medical Record Details:
Patient ID: 0x90F79bf6EB2c4f870365E785982E1f101E93b906
Doctor ID: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
Diagnosis: Influenza Type A
Prescription: Rest, increased fluid intake, and prescribed medication
Timestamp: 1746512648

Patient viewing their records...
Patient successfully retrieved their records

Testing unauthorized access...

Test 1: Unregistered doctor trying to add a record
✓ Unauthorized doctor correctly prevented from adding records

Test 2: Unregistered doctor trying to update a record
✓ Unauthorized doctor correctly prevented from updating records

Test 3: Doctor trying to update another doctor's record
Registering a second doctor...
Second doctor registered: 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC
✓ Doctor correctly prevented from updating another doctor's record

Test 4: Second doctor adding their own record
✓ Second doctor successfully added their own record
Medical record count for patient is now: 3

Test 5: Unauthorized patient trying to view records
✓ Unauthorized patient correctly prevented from viewing records

All doctor authorization tests completed successfully!
shubham@shubham:~/Desktop/bk$
```

```
Problems  Debug Console  Terminal  Ports

Doctor ID: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
Diagnosis: Seasonal Flu
Prescription: Rest, fluids, and antipyretics
Timestamp: 1746512646

Doctor updating the medical record...
Medical record updated

Updated Medical Record Details:
Patient ID: 0x90F79bf6EB2c4f870365E785982E1f101E93b906
Doctor ID: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
Diagnosis: Influenza Type A
Prescription: Rest, increased fluid intake, and prescribed medication
Timestamp: 1746512648

Patient viewing their records...
Patient successfully retrieved their records

Testing unauthorized access...

Test 1: Unregistered doctor trying to add a record
✓ Unauthorized doctor correctly prevented from adding records

Test 2: Unregistered doctor trying to update a record
✓ Unauthorized doctor correctly prevented from updating records

Test 3: Doctor trying to update another doctor's record
Registering a second doctor...
Second doctor registered: 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC
✓ Doctor correctly prevented from updating another doctor's record

Test 4: Second doctor adding their own record
✓ Second doctor successfully added their own record
Medical record count for patient is now: 3

Test 5: Unauthorized patient trying to view records
✓ Unauthorized patient correctly prevented from viewing records

All doctor authorization tests completed successfully!
shubham@shubham:~/Desktop/bk$ ^[[H
```