**PRACTICAL: 8**

**AIM:** Refer to Practical 6: Use Hardhat to compile and deploy the contract using INFURA API and Integrate the Metamask to perform transactions. Test the smart contract using the Mocha and Chai framework before deployment.

**CODE:**

**hardhat-config.js**

require("@nomicfoundation/hardhat-ethers");

require("@nomicfoundation/hardhat-toolbox");

/\*\* @type import('hardhat/config').HardhatUserConfig \*/

module.exports = {

  solidity: "0.8.20",

  defaultNetwork: "sepolia",

  // defaultNetwork: "hardhat",

  networks: {

  sepolia: {

      url: "https://eth-sepolia.g.alchemy.com/v2/xGOe71UEWL65ebXRvgj-MkL51ONfefTg",

      accounts: ["b2ba98f3f172b19f36399262d67b1d8e4d76e5a8cdb1e857cbf2a32c7b508554"]

    }

  },

  paths: {

    sources: "./contracts",

    tests: "./test",

    cache: "./cache",

    artifacts: "./artifacts"

  },

**PatientRecodTest.js**

const { expect } = require("chai");

const { ethers } = require("hardhat");

describe("PatientRecords", function () {

    let patientRecords;

    let owner;

    let nonAdmin;

    beforeEach(async function () {

        // Get signers (admin and non-admin users)

        [owner, nonAdmin] = await ethers.getSigners();

        // Deploy the contract

        const PatientRecords = await ethers.getContractFactory("PatientRecords");

        patientRecords = await PatientRecords.deploy();

        await patientRecords.waitForDeployment();

    });

    describe("Deployment", function () {

        it("Should set the right admin", async function () {

            expect(await patientRecords.admin()).to.equal(await owner.getAddress());

        });

    });

    describe("Patient Management", function () {

        it("Should allow admin to add a patient", async function () {

            const tx = await patientRecords.addPatient(1, "Alice", "Flu", "Dr. Smith", "1234567890");

            await tx.wait();  // Wait for transaction confirmation

            const patient = await patientRecords.getPatient(1);

            expect(patient.patient\_name).to.equal("Alice");

            expect(patient.disease\_type).to.equal("Flu");

            expect(patient.doctor\_name).to.equal("Dr. Smith");

            expect(patient.patient\_contact).to.equal("1234567890");

        });

        it("Should prevent non-admin from adding a patient", async function () {

            // Non-admin trying to add a patient (should fail)

            await expect(

                patientRecords.connect(nonAdmin).addPatient(2, "Bob", "Fever", "Dr. Adams", "0987654321")

            ).to.be.revertedWith("Only admin can perform this action");

        });

        it("Should prevent adding duplicate patient IDs", async function () {

            // First patient added successfully

            const tx = await patientRecords.addPatient(1, "Alice", "Flu", "Dr. Smith", "1234567890");

            await tx.wait();

            // Trying to add the same patient ID again (should fail)

            await expect(

                patientRecords.addPatient(1, "Bob", "Fever", "Dr. Adams", "0987654321")

            ).to.be.revertedWith("Patient already exists");

        });

        it("Should return all patient records", async function () {

            // Adding multiple patients

            const tx1 = await patientRecords.addPatient(1, "Alice", "Flu", "Dr. Smith", "1234567890");

            await tx1.wait();

            const tx2 = await patientRecords.addPatient(2, "Bob", "Fever", "Dr. Adams", "0987654321");

            await tx2.wait();

            const patients = await patientRecords.getAllPatients();

            // Check array length

            expect(patients.length).to.equal(2);

            // Validate first patient details

            expect(patients[0].patient\_id).to.equal(1n);

            expect(patients[0].patient\_name).to.equal("Alice");

            // Validate second patient details

            expect(patients[1].patient\_id).to.equal(2n);

            expect(patients[1].patient\_name).to.equal("Bob");

        });

        it("Should revert when getting a non-existent patient", async function () {

            await expect(

                patientRecords.getPatient(999)

            ).to.be.revertedWith("Patient not found");

        });

    });

});

**Deploy.js**

const hre = require("hardhat");

async function main() {

  const PatientRecords = await hre.ethers.getContractFactory("PatientRecords");

  const patientRecords = await PatientRecords.deploy();

  // Wait for the contract to be deployed

  await patientRecords.waitForDeployment();

  // Get the contract address

  const address = await patientRecords.getAddress();

  console.log("PatientRecords deployed to:", address);

}

main()

  .then(() => process.exit(0))

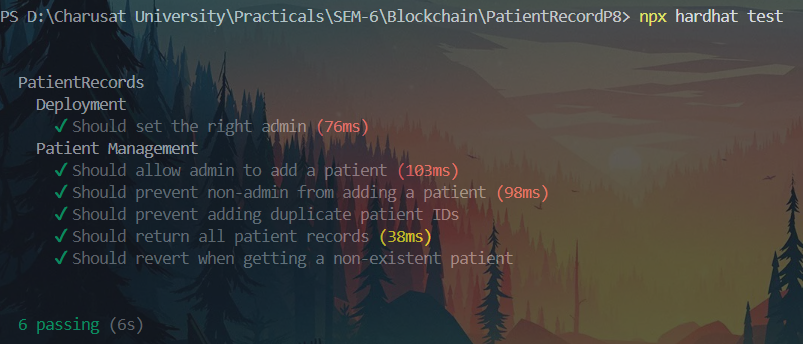
  .catch((error) => {

    console.error(error);

    process.exit(1);

  });

**OUTPUT:**





**LEARNING OUTCOME:**

Hardhat is used for compiling and deploying smart contracts on Ethereum testnets via the INFURA API. This setup allows for deploying contracts without running a local node, offering access to public Ethereum networks like Goerli or Sepolia. Contracts are thoroughly tested using Mocha and Chai, ensuring stability before deployment. Automated testing and debugging in Solidity are explored, enhancing development efficiency.