**PRACTICAL: 7**

**AIM:** Refer to Practical 5: Use Truffle to compile and deploy the contract on Ganache and Integrate the Metamask to perform transactions. Test the smart contract using the Mocha and Chai framework before deployment.

**CODE:**

**Truffle-config.js**

const HDWalletProvider = require('@truffle/hdwallet-provider');

const mnemonic = "wet food text simple cloth glow economy usual ready dune gravity crash";

module.exports = {

  networks: {

    development: {

      host: "127.0.0.1",

      port: 8545,

      network\_id: "\*"

    },

    ganache: {

      provider: () => new HDWalletProvider(mnemonic, "http://127.0.0.1:8545"),

      network\_id: "\*"

    }

  },

  mocha: {

    timeout: 100000

  },

  compilers: {

    solc: {

      version: "0.8.20"

    }

  }

};

**CertificateTest.js**

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

contract("EventCertificate", (accounts) => {

  let EventCertificate;

  let contractInstance;

  const admin = accounts[0];

  const nonAdmin = accounts[1];

  before(async () => {

    EventCertificate = artifacts.require("EventCertificate");

    contractInstance = await EventCertificate.new({ from: admin });

  });

  it("Should set the deployer as the admin", async () => {

    const contractAdmin = await contractInstance.admin();

    expect(contractAdmin).to.equal(admin);

  });

  it("Should issue a new certificate by admin", async () => {

    const certificateId = 1;

    const studentName = "John Doe";

    const branch = "Computer Science";

    const courseName = "Blockchain Development";

    const grade = "A+";

    const tx = await contractInstance.issueCertificate(

      certificateId,

      studentName,

      branch,

      courseName,

      grade,

      { from: admin }

    );

    expect(tx.logs[0].event).to.equal("CertificateIssued");

    expect(tx.logs[0].args.studentName).to.equal(studentName);

  });

  it("Should NOT allow non-admin to issue a certificate", async () => {

    try {

      await contractInstance.issueCertificate(

        2,

        "Alice",

        "Electrical Engineering",

        "Smart Grid",

        "A",

        { from: nonAdmin }

      );

      expect.fail("Non-admin was able to issue a certificate");

    } catch (error) {

      expect(error.message).to.include("Only admin can perform this action");

    }

  });

  it("Should retrieve certificate details", async () => {

    const certificateId = 1;

    const certDetails = await contractInstance.getCertificate(certificateId);

    expect(certDetails[0].toNumber()).to.equal(certificateId);

    expect(certDetails[1]).to.equal("John Doe");

    expect(certDetails[2]).to.equal("Computer Science");

    expect(certDetails[3]).to.equal("Blockchain Development");

    expect(certDetails[4]).to.equal("A+");

  });

  it("Should verify a certificate correctly", async () => {

    const certificateId = 1;

    const isValid = await contractInstance.verifyCertificate(

      certificateId,

      "John Doe",

      "Computer Science",

      "Blockchain Development",

      "A+"

    );

    expect(isValid).to.be.true;

  });

  it("Should return false for incorrect certificate verification", async () => {

    const certificateId = 1;

    const isValid = await contractInstance.verifyCertificate(

      certificateId,

      "John Doe",

      "Mechanical Engineering",

      "Blockchain Development",

      "A+"

    );

    expect(isValid).to.be.false;

  });

  it("Should not retrieve non-existent certificate", async () => {

    try {

      await contractInstance.getCertificate(99);

      expect.fail("Should have thrown an error");

    } catch (error) {

      expect(error.message).to.include("revert");

      expect(error.message).to.include("Certificate not found");

    }

  });

});

**Deploy.js**

const EventCertificate = artifacts.require("EventCertificate");

module.exports = async function (deployer, network, accounts) {

  await deployer.deploy(EventCertificate);

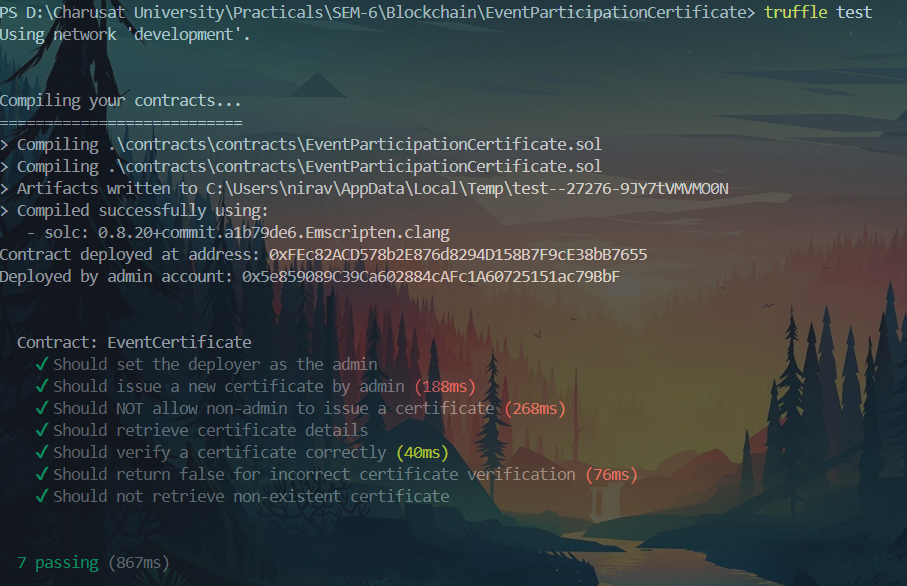
  const contractInstance = await EventCertificate.deployed();

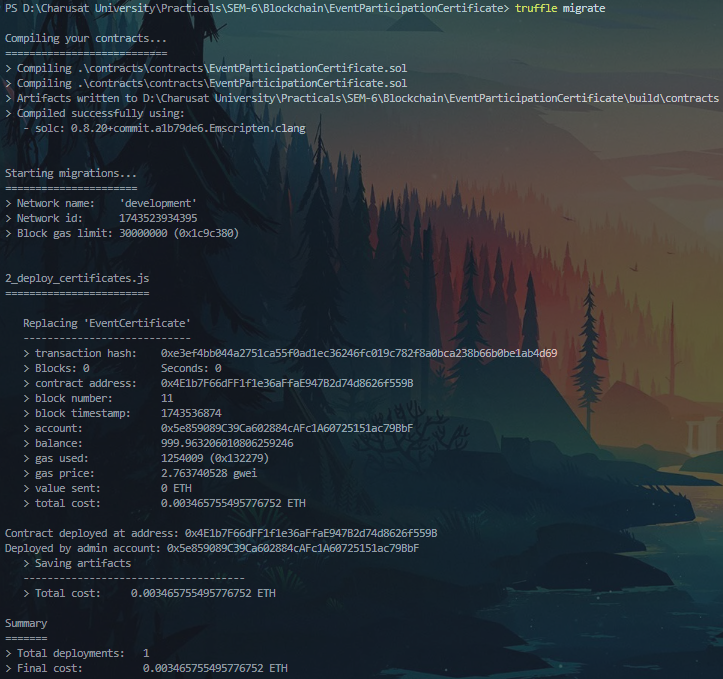
  console.log(`Contract deployed at address: ${contractInstance.address}`);

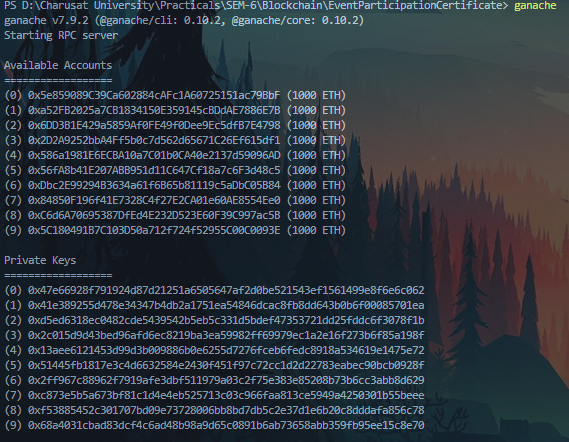
  console.log(`Deployed by admin account: ${accounts[0]}`);

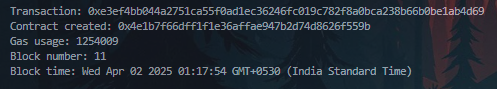
};

**OUTPUT:**



****

****

****

**LEARNING OUTCOME:**

The Truffle framework is used to compile, deploy, and test smart contracts on a local Ethereum network set up with Ganache. Truffle automates contract compilation, migration, and interaction. Mocha and Chai are used for testing contracts, ensuring that they work as expected before deployment. Metamask is integrated for secure blockchain transactions, providing a seamless connection to the local network.