Blockchain & Solidity Lab2 – Voting dApp Development

S2BC

Lab 2: Testing Ethereum Smart Contracts with Hardhat

- BUILD / **TEST** / INTEGRATE / RUN

Objective: In this lab, we will focus on testing Ethereum Smart Contracts using Hardhat. Testing is a crucial step in the development process to ensure the reliability and security of your smart contracts.

Table of Contents

1. **Introduction to Testing Ethereum Smart Contracts**
2. **The Contract Testing Object**
3. **Running the Tests**
4. **Best Practices for Smart Contract Testing**
5. **Integration with Hardhat**

Introduction to Testing Ethereum Smart Contracts

In this section, we'll explore the importance of testing smart contracts and how it ensures the integrity of the blockchain application. Testing helps identify and fix potential vulnerabilities in your code before deployment.

The Contract Testing Object

The testing object in Solidity is a critical component for writing comprehensive test cases. It allows you to simulate various scenarios and interactions with your smart contracts to ensure they function as intended.

Running the Tests

To run tests using Hardhat, follow these steps:

1. Ensure you have Hardhat installed in your development environment.
2. Create a new directory for your tests and write test files using the testing object discussed earlier.
3. Use the Hardhat command-line interface (CLI) to execute the tests.
4. Review the output for any failed tests and debug accordingly.

Best Practices for Smart Contract Testing

Writing effective test cases is crucial for contract security. Here are some best practices to consider:

- Use descriptive test case names to clearly indicate the purpose of each test.
- Write assertions to validate contract behavior. This ensures that contracts function as expected.
- Test edge cases and potential failure scenarios to cover all possible outcomes.

Integration with Hardhat

Integrating Hardhat into your development workflow is straightforward and highly beneficial for efficient testing. Follow these steps:

1. Add Hardhat as a development dependency in your project.
2. Configure your Hardhat environment to suit your specific needs, including network configurations and plugins.
3. Write tests using the testing object and run them using the Hardhat CLI.

Example of test

```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("Voting Contract", function () {
  it("Non-owner should not be allowed to start an election", async function () {
    // Deploy the Voting contract
    const Voting = await ethers.getContractFactory("Voting");
    const votingInstance = await Voting.deploy();
    await votingInstance.deployed();

    // Connect to a wallet to simulate a non-owner address
    const [nonOwner] = await ethers.getSigners();

    // Try to start the election
    try {
      await votingInstance.connect(nonOwner).startElection(["Candidate 1", "Candi
      // If the above line doesn't throw an error, the test should fail
      expect(true).to.equal(false); // This line should not be reached
    } catch (error) {
      // Check if the error message contains the expected error message
      expect(error.message).to.contain("not authorized to start election");
    }
  });
});
```

Contact

S2BC