Centurion University
Shaping Lives... Empowering Communities...

School: ..... Campus: .....

Academic Year: ..... Subject Name: ..... Subject Code: .....

Semester: ..... Program: ..... Branch: ..... Specialization: .....

Date: .....

Applied and Action Learning

(Learning by Doing and Discovery)

Name of the Experiement : Contract QA – Testing Smart Contracts

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

ALGORITHM:

1.Write or import a Solidity smart contract (e.g., SimpleStorage.sol).
2.Initialize a project using Hardhat or Truffle.
3.Write test scripts using JavaScript or TypeScript to test smart contract functions:
-Deploy the contract
-Call read/write functions
-Test failure scenarios (e.g., require checks)
4.Run tests on a local blockchain (like Hardhat Network or Ganache).
5.View test results and debug any failures or incorrect behaviors.
6.Fix issues in the smart contract and re-run tests until all pass.

* Software used

1.Remix IDE
2.Hardhat
3.Node.js
4.MetaMask
5.Mocha/Chai for testing.

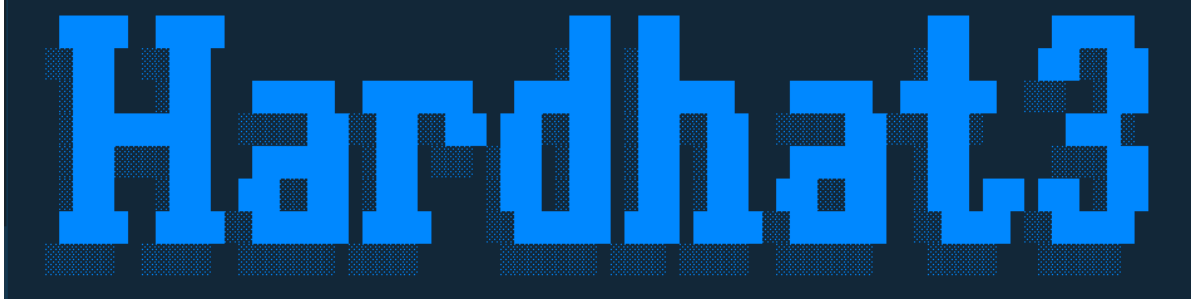
*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Testing Phase: Compilation of Code (error detection)

1.Create a Hardhat project

```
D:\BLOCKCHAIN\BLOCKCHAIN-LABS\hfgdj\testing>npm install --save-dev hardhat

added 59 packages, and audited 60 packages in 18s

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

D:\BLOCKCHAIN\BLOCKCHAIN-LABS\hfgdj\testing>
```

```
D:\BLOCKCHAIN\BLOCKCHAIN-LABS\hfgdj\testing>npx hardhat --init
```



2.Write a small smart contract

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

contract SimpleStorage {
    uint public value;

    constructor(uint _value) {
        value = _value;
    }

    function setValue(uint _value) public {
        value = _value;
    }
}
```

3.Write test cases for it

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

describe("SimpleStorage", function () {
  it("Should deploy and return the initial value", async function () {
    const SimpleStorage = await ethers.getContractFactory("SimpleStorage");
    const simpleStorage = await SimpleStorage.deploy(100);
    await simpleStorage.deployed();

    expect(await simpleStorage.value()).to.equal(100);
  });
```


*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)


```
  it("Should update the value", async function () {
    const SimpleStorage = await ethers.getContractFactory("SimpleStorage");
    const simpleStorage = await SimpleStorage.deploy(100);
    await simpleStorage.deployed();

    await simpleStorage.setValue(200);
    expect(await simpleStorage.value()).to.equal(200);
  });
});
```

SimpleStorage
✔ Should deploy and return the initial value (500ms)
✔ Should update the value (300ms)

2 passing (1s)

* Observations

1.Automated testing ensures smart contract functionality is correct before deployment to production.
2.Unit tests help catch bugs or logical issues early in the development cycle.
3.Using test frameworks like Hardhat improves reliability and reduces manual testing.
4.Smart contract testing saves gas and reduces risk of security vulnerabilities.

ASSESSMENT

Rubrics	Full Mark	Marks Obtained	Remarks
Concept	10		
Planning and Execution/ Practical Simulation/ Programming	10		
Result and Interpretation	10		
Record of Applied and Action Learning	10		
Viva	10		
Total	**50**		

Signature of the Student:

Name :

Regn. No. :

Signature of the Faculty:

**As applicable according to the experiment.*
Two sheets per experiment (10-20) to be used.