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

Objective/Aim:

To learn how to test and verify the functionality of smart contracts by simulating deployment, transaction execution, and error handling using Remix IDE, Solidity, and MetaMask on an Ethereum test network.

Apparatus/Software Used:

- Remix IDE
- Solidity Compiler (v0.8.24)
- MetaMask Wallet (for deployment)
- Ethereum Testnet (e.g., Sepolia)

Theory/Concept:

what is smart contract

A smart contract is a blockchain-based digital agreement that automatically runs when specific conditions are met. It eliminates third-party involvement by enforcing rules directly through code, making transactions faster, more secure, and transparent.

Contract QA (Quality Assurance)

Smart contract QA is the process of verifying the correctness, reliability, and efficiency of a contract before it is deployed to the blockchain. Since contracts are immutable after deployment, testing helps prevent vulnerabilities and logical errors.

Main Aspects of Smart Contract QA:

1. **Unit Testing:**
   Each function is tested separately to ensure expected outcomes.
2. **Integration Testing:**
   Ensures all functions and modules work together properly.
3. **Error and Exception Handling:**
   Checks that invalid inputs or unauthorized access trigger proper error messages.
4. **Gas Optimization:**
   Reviews contract code to minimize gas consumption during transactions.

Proper QA ensures that the smart contract runs smoothly in live environments and protects users from potential losses due to coding mistakes or exploits.

Procedure:

Step 1: Open Remix IDE — go to **https://remix.ethereum.org**..
Step 2: Create a new file MyContract.sol.
Step 3: Write a simple contract (token or voting). Save the file.
Step 4: *In* **Solidity Compiler**, choose **v0.8.x**.
Step 5: Click **Compile MyContract.sol**.
Step 6: Install/open **MetaMask** browser extension.
Step 7: Connect MetaMask to a testnet (Goerli or Sepolia) and fund with test ETH.
Step 8: In Remix → **Deploy & Run Transactions**, set **Environment = Injected Provider (MetaMask)**.
Step 9: Click **Deploy** and confirm the MetaMask transaction.
Step 10: Run functional tests: call contract functions (mint, transfer, vote, etc.).
Step 11: Watch Remix console & MetaMask for transaction hashes and confirmations.
Step 12: Test error cases (unauthorized calls, bad inputs) to ensure proper `reverts`.
Step 13: Record results: expected vs actual, transaction hash, and status (pass/fail).

Observation Table:

Test Case	Function Tested	Expected Result	Actual Result	Status
1	createRecord()	Record created successfully	✅ Success	Pass
2	transferOwnership()	Ownership transferred correctly	✅ Success	Pass
3	Invalid Access	Unauthorized access reverted	❌ Revert Error	Pass
4	updateData()	Data updated securely	✅ Success	Pass

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 Faculty:

Signature of the Student:

Name :
Regn. No.