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 :

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

Algorithm:

1.Open Remix IDE and write the SimpleStorage.sol smart contract.
2.Compile the smart contract using the Solidity compiler in Remix.
3.Copy the generated ABI after successful compilation.
4.Deploy the contract to the Sepolia Testnet using MetaMask.
5.Copy the deployed contract address.
6.Create a React frontend project using create-react-app.
7.Add the contract address and network information to the .env file.
8.Install web3.js to interact with the blockchain.
9.Use the ABI and contract address to connect the frontend with the smart contract.
10.Design the UI in App.js using Web3.js to store and retrieve data.

* Softwares used

1.MetaMask Wallet
2.Remix IDE
3.Brave browser

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

* Testing Phase: Compilation of Code (error detection)

Go to remix ide and write a smart contract on simplestorage.sol and compile it

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

contract SimpleStorage {
    uint public storedData;

    constructor(uint _data) {
        storedData = _data;
    }

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

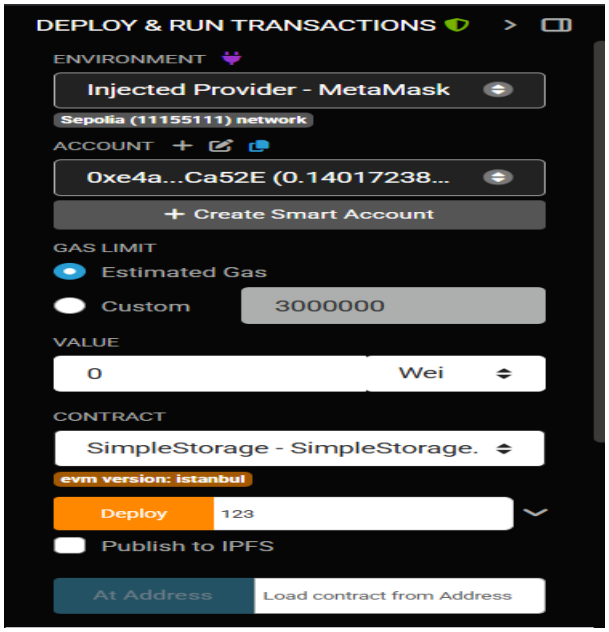
After compile the smart contract there is a ABI of the smart contract

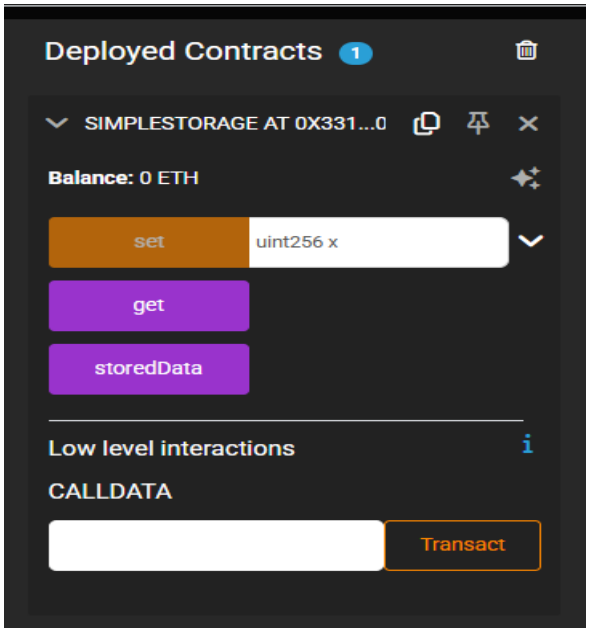
```json
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "x",
            "type": "uint256"
        }
    ],
    "name": "set",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "_data",
            "type": "uint256"
        }
    ],
    "stateMutability": "nonpayable",
    "type": "constructor"
},
{
    "inputs": [],
    "name": "get",
    "outputs": [
        {
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "storedData",
    "outputs": [
        {
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
        }
    ],
    "stateMutability": "view",
    "type": "function"
}
]
;
```

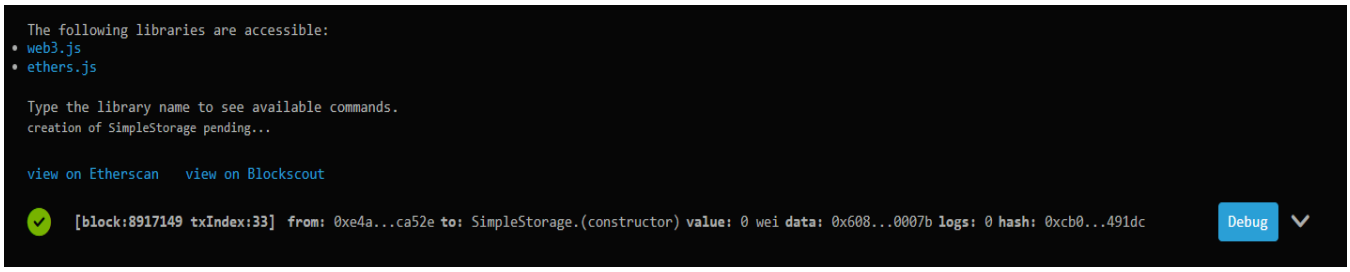

*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Testing Phase: Compilation of Code (error detection)

After compilation ,deploy the smart contract and choose the enviornment as injector provider-metamask then give some value and start deploy







In this Smart contract we have two accessible libraries one is ether.js and another is web3.js we have to work on web3.js

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

* Implementation Phase: Final Output (no error)

Now we have to work on frontend first create a folder for your frontend then open terminal to install the react modules . Then create a ABI.js file inside your src folder where we have to store the abi of our smart contract and then create a .env file in the root of the project folder to dtore contract address and tectnet network

frontend > .env

```
REACT_APP_CONTRACT_ADDRESS=0x3b3e221F2aE6894Da182fbA1D93Ae066CC7fCd55
REACT_APP_NETWORK=sepolia
```

Now in App.js write your frontend code and wallet connection code importing web3.

App.js M | App.test.js | index.js | reportWebVitals.js | setupTests.js | .env U | abi.js U

frontend > src > App.js > App

```
import React, { useState } from 'react';
import Web3 from 'web3';
import { simpleStorageABI } from './abi';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import { FaSpinner, FaWallet, FaDatabase, FaSync, FaPlug, FaSignOutAlt } from 'react-icons/fa';

const contractAddress = process.env.REACT_APP_CONTRACT_ADDRESS;

function App() {
  const [account, setAccount] = useState(null);
  const [contract, setContract] = useState(null);
  const [storedValue, setStoredValue] = useState(null);
  const [inputValue, setInputValue] = useState('');
  const [loading, setLoading] = useState(false);

  const connectWallet = async () => {
    if (window.ethereum) {
      try {
        const web3Instance = new Web3(window.ethereum);
        await window.ethereum.request({ method: 'eth_requestAccounts' });
        const accounts = await web3Instance.eth.getAccounts();
        const userAccount = accounts[0];

        const contractInstance = new web3Instance.eth.Contract(simpleStorageABI, contractAddress);

        setAccount(userAccount);
        setContract(contractInstance);

        toast.success("Wallet connected!");
        fetchStoredValue(contractInstance);
      } catch (err) {
        toast.error("Connection failed.");
        console.error(err);
      }
    } else {
      toast.error("Please install MetaMask.");
    }
  };
```


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

* Implementation Phase: Final Output (no error)

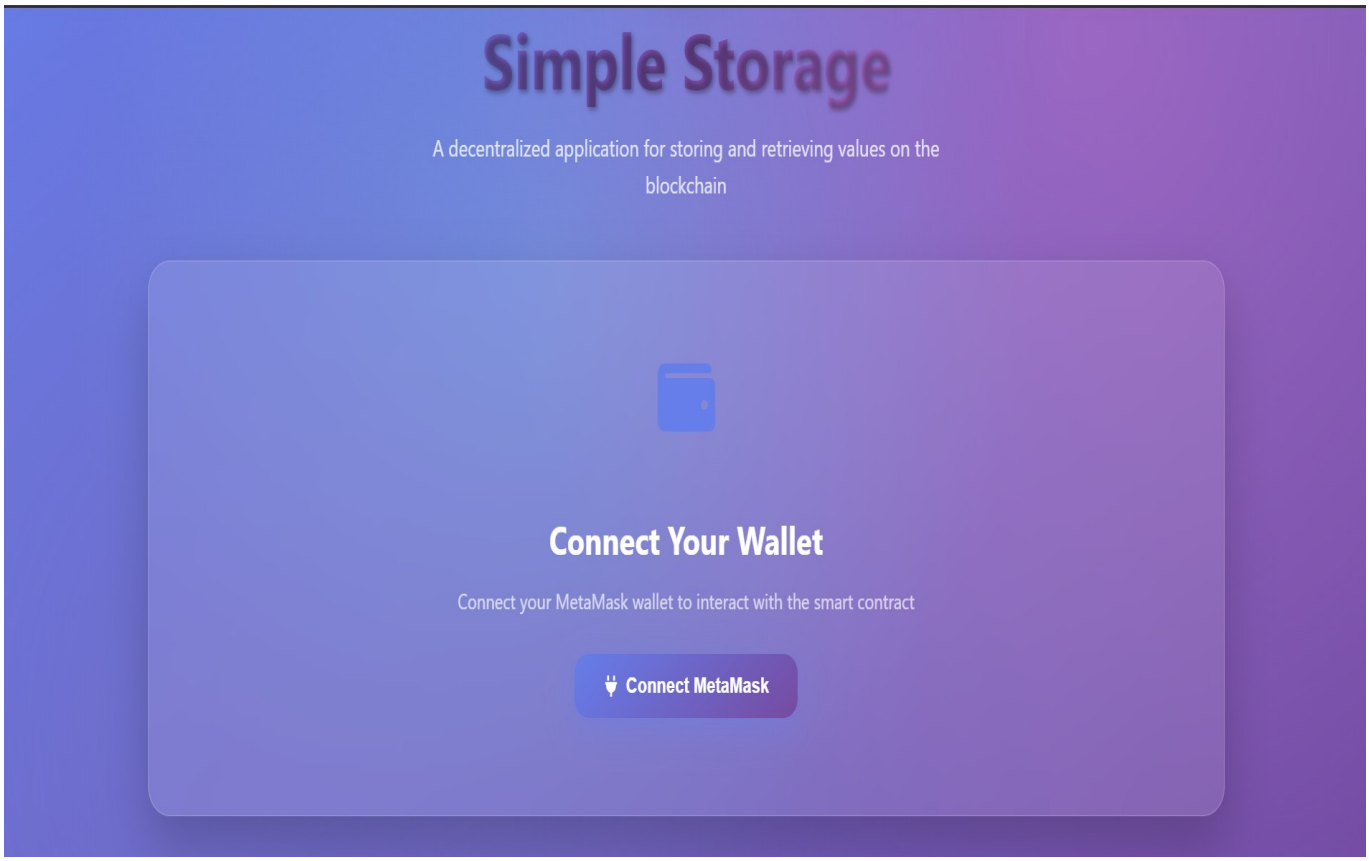
```
10   function App() {
41     const disconnectWallet = () => {
42       setAccount(null);
43       setContract(null);
44       setStoredValue(null);
45       toast.info("Wallet disconnected.");
46     };
47
48     const fetchStoredValue = async (contractRef = contract) => {
49       try {
50         const value = await contractRef.methods.get().call();
51         setStoredValue(value.toString());
52       } catch (err) {
53         toast.error("Failed to fetch data.");
54         console.error(err);
55       }
56     };
57
58     const handleSet = async () => {
59       if (contract && inputValue) {
60         try {
61           setLoading(true);
62           await contract.methods.set(inputValue).send({ from: account });
63           setInputValue('');
64           toast.success("Value updated successfully!");
65           fetchStoredValue();
66         } catch (err) {
67           toast.error("Transaction failed.");
68           console.error(err);
69         } finally {
70           setLoading(false);
71         }
72       }
73     };
```

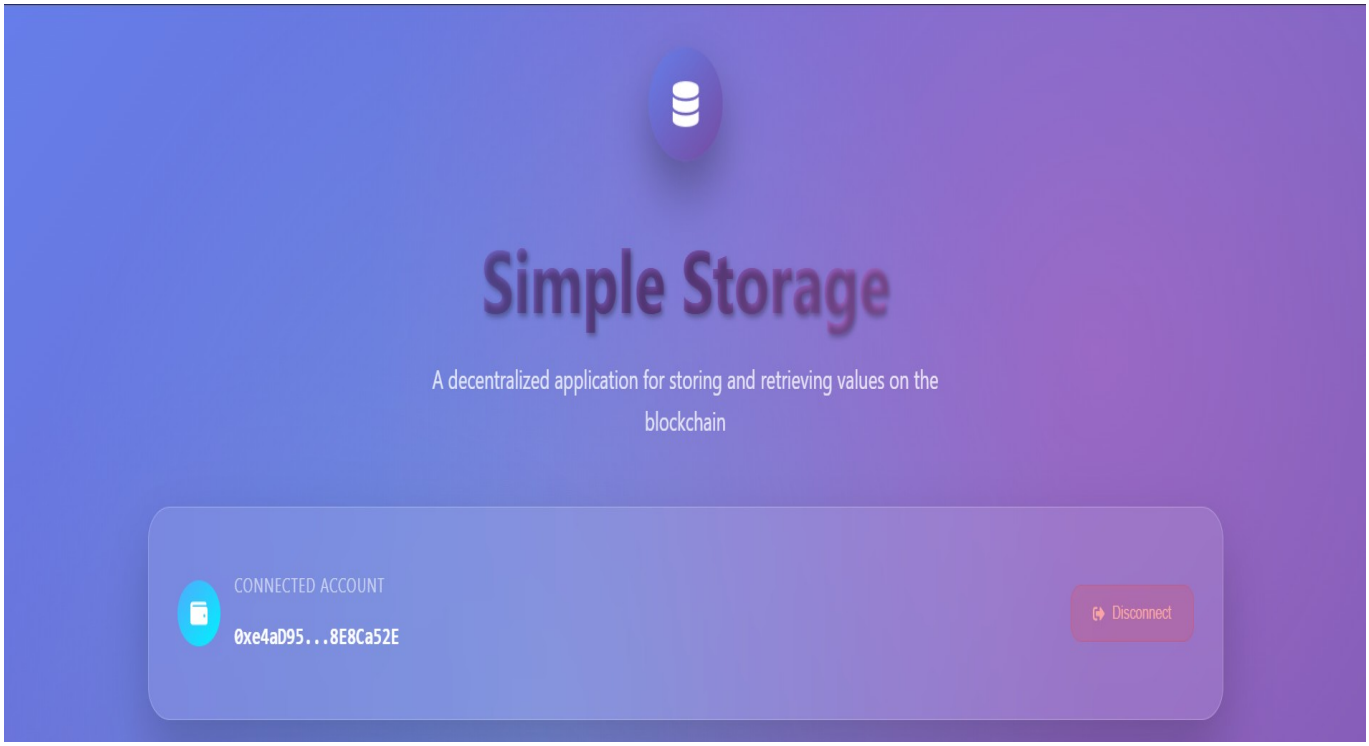
After write all the coder now for frontend design write the css .after writing all coder now install the web3 packages inside your frontend folder to install all the packages of web3 the command is -npm install web3
after installing all the packages now to run the frontend write the command
npm start

```
PROBLEMS   OUTPUT   TERMINAL   PORTS

You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://10.114.46.119:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```


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

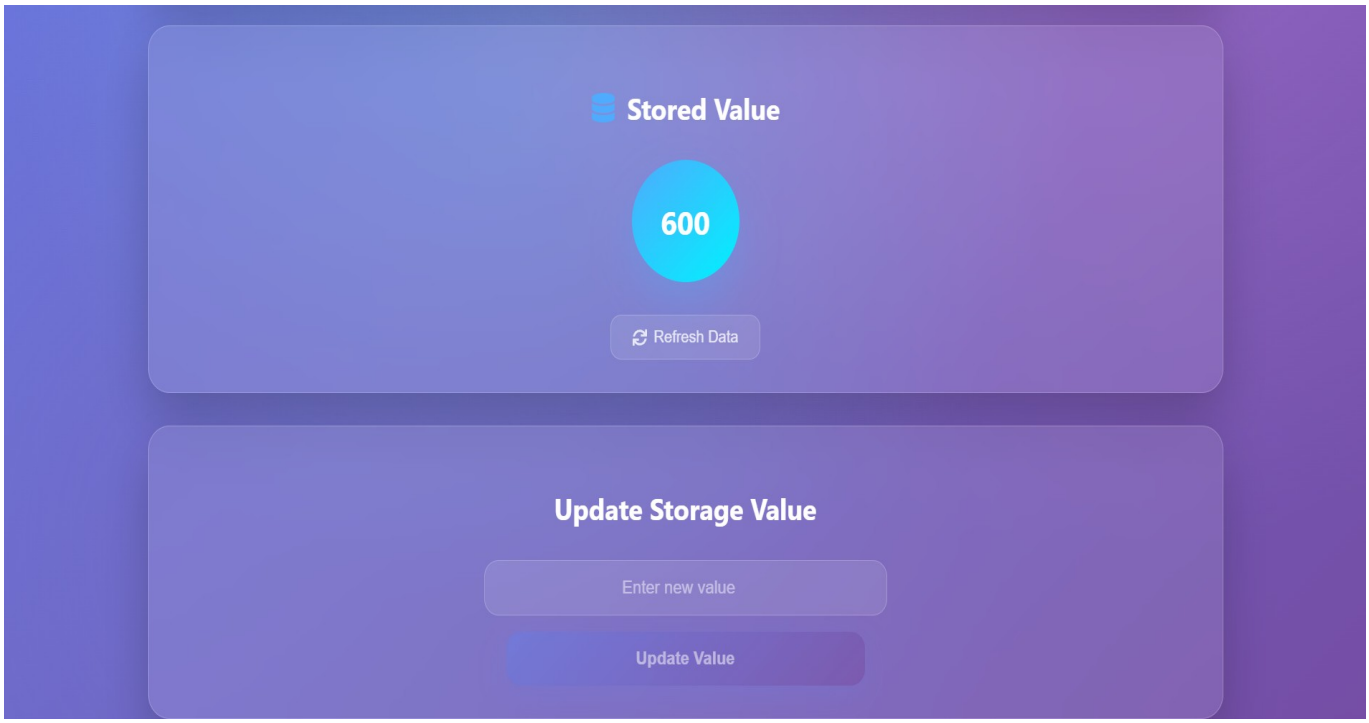
* Implementation Phase: Final Output (no error)





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

* Implementation Phase: Final Output (no error)



Now in this frontend you can update value and check stored value

* Observations

1.Writing and deploying a smart contract on Remix IDE is efficient and user-friendly for beginners.
2.Web3.js effectively enables communication between the React frontend and the Ethereum blockchain.
3.The integration of contract ABI and address in the frontend allows dynamic interaction with the deployed contract.

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.