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 : UI for DApps – Building a DApp Frontend

* 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 ether.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

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Counter {
    uint public count;
    constructor(uint _start) {    Infinite gas 132000 gas
        count = _start;
    }
    function increment() public {    Infinite gas
        count += 1;
    }
    function decrement() public {    Infinite gas
        require(count > 0, "Counter is already at zero");
        count -= 1;
    }
    function getCount() public view returns (uint) {    2453 gas
        return count;
    }
}
```

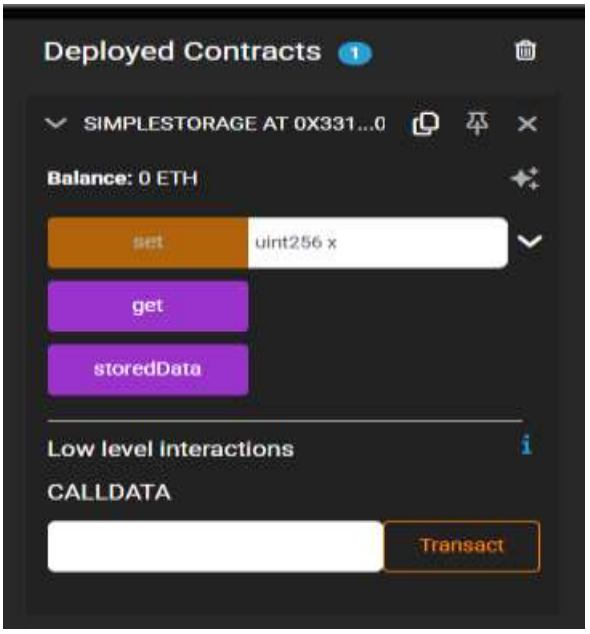
After compile the smart contract there is a ABI of the smart contract


```
[
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "_start",
                "type": "uint256"
            }
        ],
        "stateMutability": "nonpayable",
        "type": "constructor"
    },
    {
        "inputs": [],
        "name": "count",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "decrement",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "getCount",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "increment",
        "outputs": [],
        "stateMutability": "nonpayable",
        "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 ether.js

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

* Implementation Phase: Final Output (no error)

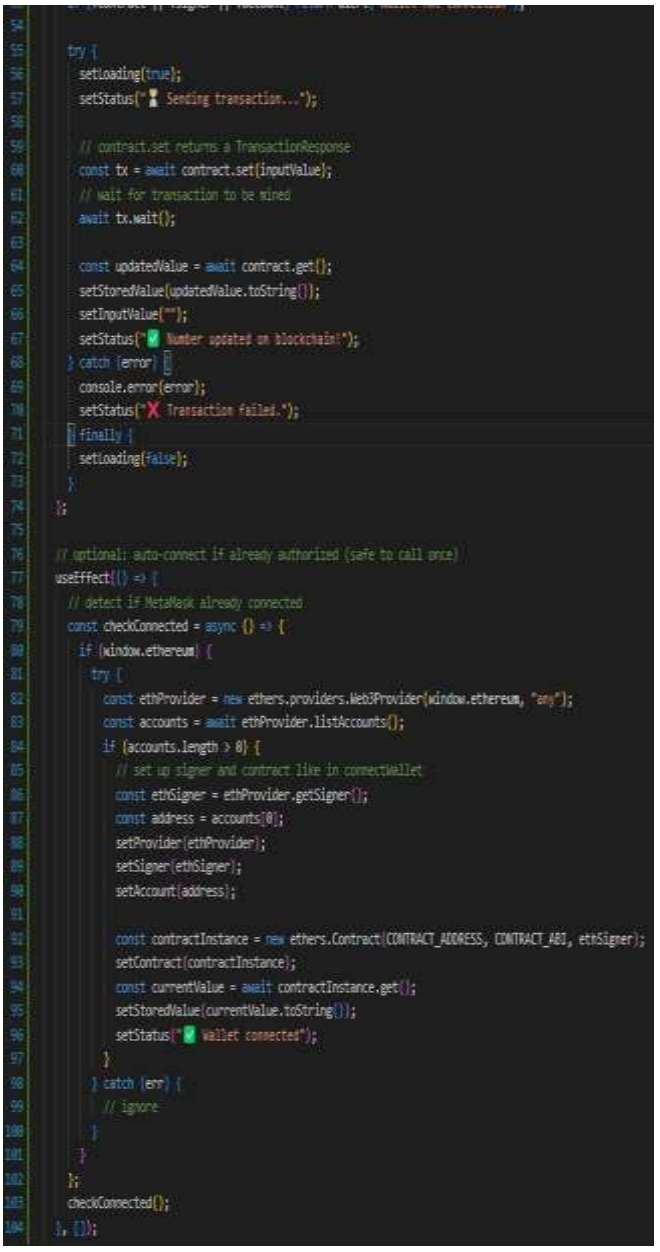
Now we have to work on frontend first create a folder for your frontend then open terminal to install the react modules . Then create a ABI.js file inside your src folder where we have to store the abi of our smart contract and then create a .env file in the root of the project folder to dtore contract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

simple-storage-dapp > src > .env

```
REACT_APP_CONTRACT_ADDRESS=0xa62463A56EE9D742F810920F56cEbc4B696eBd0a
REACT_APP_NETWORK=sepolia
```

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

```
import React, { useEffect, useState } from "react";
import { ethers } from "ethers";
import { CONTRACT_ABI } from "./abi";

// 🔗 Replace with your deployed contract address from Remix
const CONTRACT_ADDRESS = "0xa62463A56EE9D742F810920F56cEbc4B696eBd0a";

function App() {
  const [provider, setProvider] = useState(null);
  const [signer, setSigner] = useState(null);
  const [account, setAccount] = useState("");
  const [contract, setContract] = useState(null);
  const [inputValue, setInputValue] = useState("");
  const [storedValue, setStoredValue] = useState("");
  const [loading, setLoading] = useState(false);
  const [status, setStatus] = useState("");

  // 🦊 Connect Wallet (ethers.js)
  const connectWallet = async () => {
    if (!window.ethereum) {
      alert("Please install MetaMask to use this DApp.");
      return;
    }

    try {
      const ethProvider = new ethers.providers.Web3Provider(window.ethereum, "any");
      // prompt user for account connections
      await ethProvider.send("eth_requestAccounts", []);
      const ethSigner = ethProvider.getSigner();
      const address = await ethSigner.getAddress();

      setProvider(ethProvider);
      setSigner(ethSigner);
      setAccount(address);

      const contractInstance = new ethers.Contract(CONTRACT_ADDRESS, CONTRACT_ABI, ethSigner);
      setContract(contractInstance);

      // read stored value (may be BigNumber)
      const currentValue = await contractInstance.get();
      setStoredValue(currentValue.toString());

      setStatus("✅ Wallet connected");
    } catch (error) {
      console.error("Wallet connection error:", error);
      setStatus("❌ Failed to connect wallet.");
    }
  };

  // 📝 Handle Submit (ethers.js)
  const handleSubmit = async () => {
    if (!inputValue) return alert("Please enter a number.");
    if (!contract || !signer || !account) return alert("Wallet not connected.");
```



```
    try {
      setLoading(true);
      setStatus("⏳ Sending transaction...");

      // contract.set returns a TransactionResponse
      const tx = await contract.set(inputValue);
      // wait for transaction to be mined
      await tx.wait();

      const updatedValue = await contract.get();
      setStoredValue(updatedValue.toString());
      setInputValue("");
      setStatus("✅ Number updated on blockchain!");
    } catch (error) {
      console.error(error);
      setStatus("❌ Transaction failed.");
    } finally {
      setLoading(false);
    }
  };

  // optional: auto-connect if already authorized (safe to call once)
  useEffect(() => {
    // detect if MetaMask already connected
    const checkConnected = async () => {
      if (window.ethereum) {
        try {
          const ethProvider = new ethers.providers.Web3Provider(window.ethereum, "any");
          const accounts = await ethProvider.listAccounts();
          if (accounts.length > 0) {
            // set up signer and contract like in connectWallet
            const ethSigner = ethProvider.getSigner();
            const address = accounts[0];
            setProvider(ethProvider);
            setSigner(ethSigner);
            setAccount(address);

            const contractInstance = new ethers.Contract(CONTRACT_ADDRESS, CONTRACT_ABI, ethSigner);
            setContract(contractInstance);
            const currentValue = await contractInstance.get();
            setStoredValue(currentValue.toString());
            setStatus("✅ Wallet connected");
          }
        } catch (err) {
          // ignore
        }
      }
    };
    checkConnected();
  }, []);
```


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

* Implementation Phase: Final Output (no error)

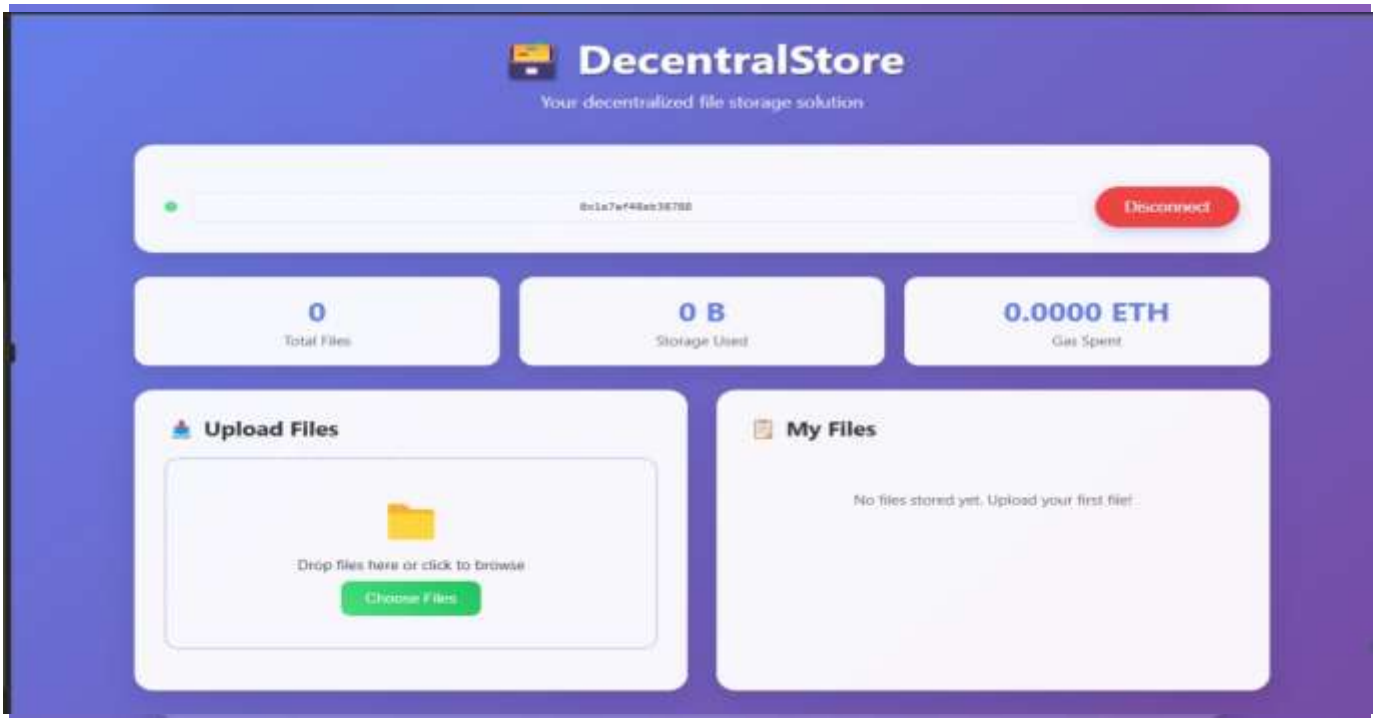
```
return (
  <div style={{ padding: "40px", fontFamily: "Arial", maxWidth: "600px", margin: "auto" }}>
    <h2>🦊 Simple Storage DApp (ethers.js)</h2>

    {!account ? (
      <button
        onClick={connectWallet}
        style={{
          padding: "10px 20px",
          backgroundColor: "#007bff",
          color: "white",
          border: "none",
          borderRadius: "5px",
          marginBottom: "20px",
        }}
      >
        🔌 Connect Wallet
      </button>
    ) : (
      <div style={{ marginBottom: "20px" }}>
        <p><strong>🔴 Wallet:</strong> {account}</p>
        <p><strong>🟡 Stored Number:</strong> {storedValue}</p>
      </div>
    )}

    {account && (
      <>
        <div>
          <input
            type="number"
            value={inputValue}
            onChange={(e) => setInputValue(e.target.value)}
            placeholder="Enter a number"
            style={{ padding: "8px", width: "70%", marginRight: "5px" }}
            disabled={loading}
          />
          <button onClick={handleSubmit} disabled={loading}>
            {loading ? "⏳ Updating..." : "Set Number"}
          </button>
        </div>

        {status && (
          <p style={{ marginTop: "20px", color: status.includes("✅") ? "green" : "red" }}>
            {status}
          </p>
        )}
      </>
    )}
  </div>
);
}

export default App;
```

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

```
> vite


  VITE v7.0.6  ready in 1510 ms

  ➜ Local:   http://localhost:5173/

  VITE v7.0.6  ready in 1510 ms
```


*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. ether.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.*