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 : Frontend Connect – Web3.js Integration

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

ALGORITHM:

- **Start** by setting up the development environment with **Node.js**, **npm**, and a code editor (e.g., VS Code).
- **Install Web3.js library** using npm to enable interaction between the frontend and Ethereum blockchain.
- **Connect to the blockchain** by creating a Web3 instance and linking it to a local blockchain (Ganache) or test network (Ropsten, Rinkeby) via provider or MetaMask.
- **Load the smart contract ABI and address** in the frontend application to interact with deployed contracts.
- **Write frontend functions** to call contract methods for reading (e.g., balanceOf) and writing (e.g., transfer) data.
- **Test transactions and data retrieval** from the frontend, verifying successful communication with the blockchain.
- **End** the process after confirming that the frontend can interact with the smart contract correctly and display real-time blockchain data.

* Software used

1. Metamask wallet
2. Remix IDE
3. Brave Browser

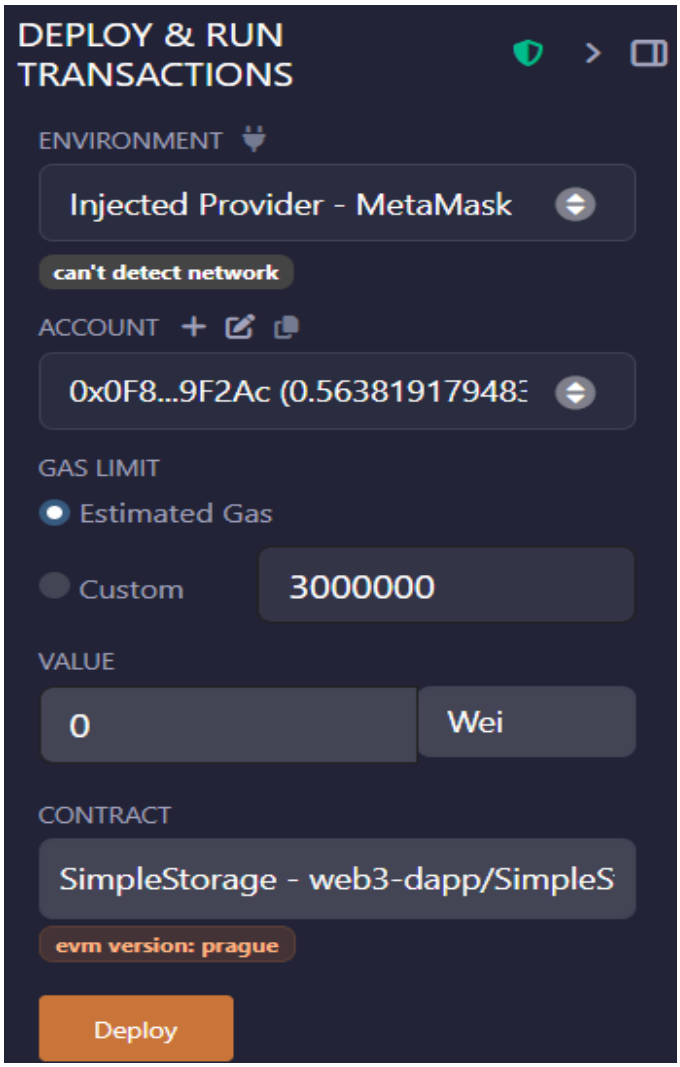
*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

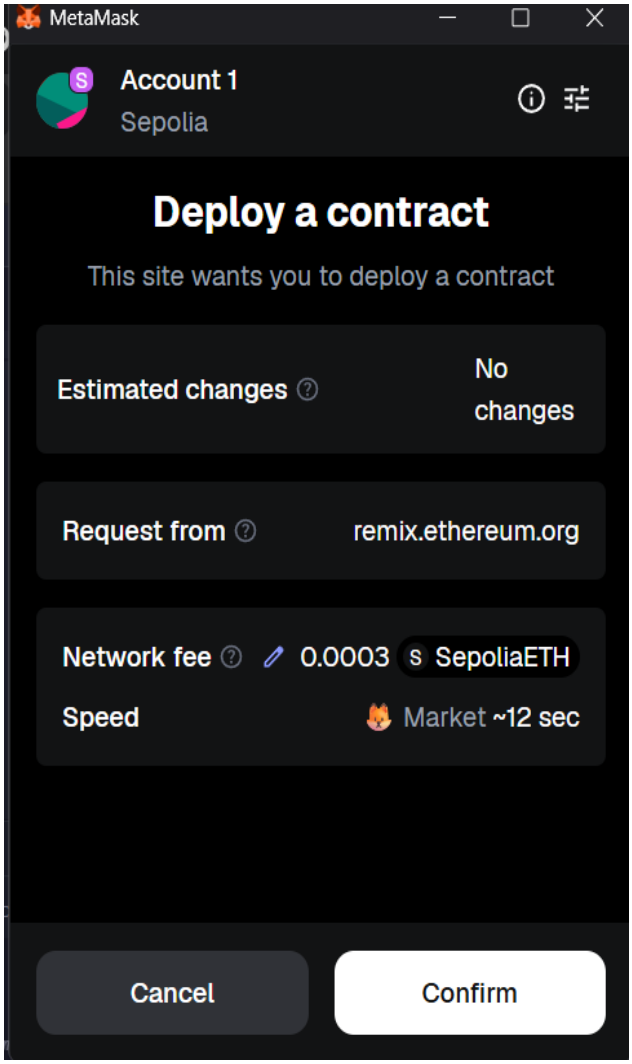
* Testing Phase: Compilation of Code (error detection)

Smart contract solidity code

```
//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 compilation deploy the smart contract in sepolia test network using metamask

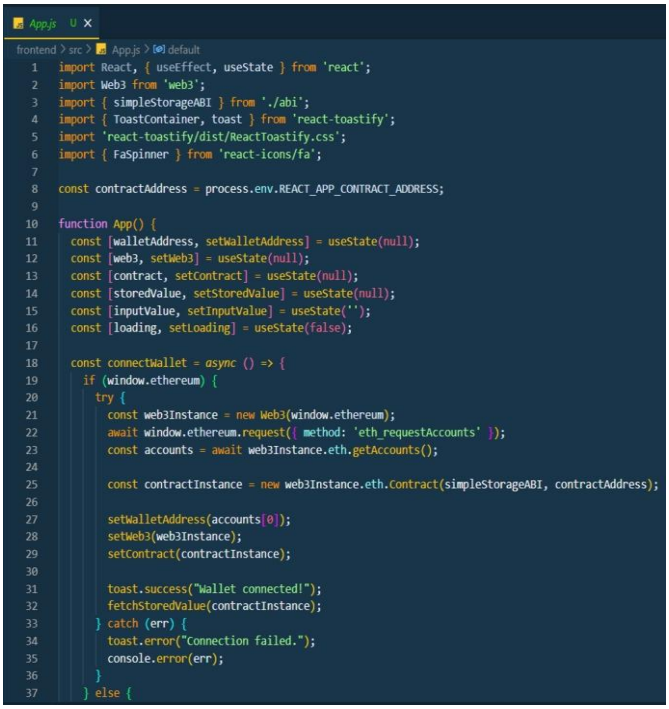




*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)

Now we have to work on our frontend first create a folder for your frontend then open terminal and install react modules needed for the project. Now create Abi.js file in the src folder where we have to store the abi for our smart contract and now create a .env file and store the contract address and network information.



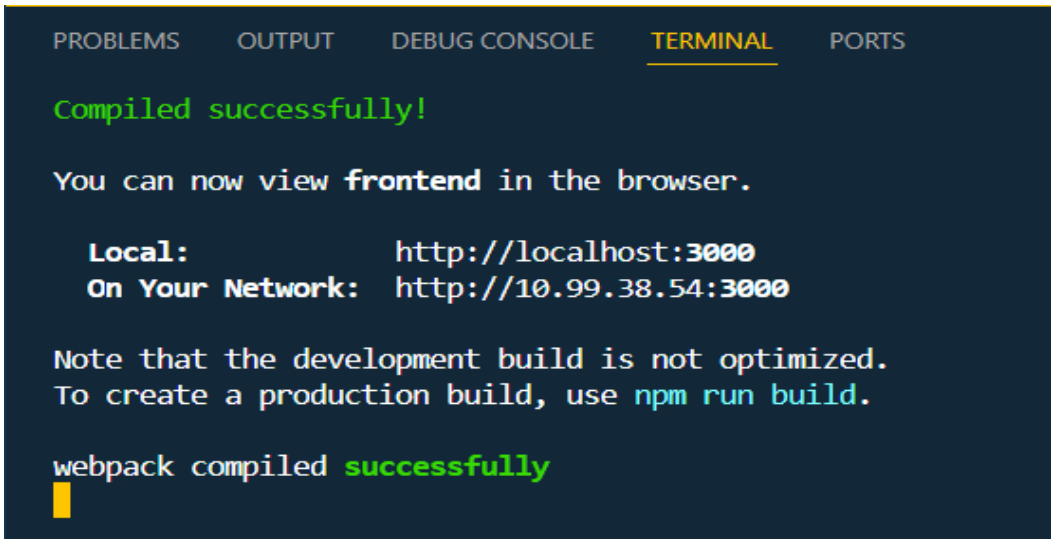
```
frontend > src > App.js > default
import React, { useEffect, useState } from 'react';
import Web3 from 'web3';
import { simpleStorageABI } from './abi';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import { FaSpinner } from 'react-icons/fa';

const contractAddress = process.env.REACT_APP_CONTRACT_ADDRESS;

function App() {
  const [walletAddress, setWalletAddress] = useState(null);
  const [web3, setWeb3] = 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 contractInstance = new web3Instance.eth.Contract(simpleStorageABI, contractAddress);

        setWalletAddress(accounts[0]);
        setWeb3(web3Instance);
        setContract(contractInstance);

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

```
      toast.error("Please install MetaMask.");
    }
  };

  const disconnectWallet = () => {
    setWalletAddress(null);
    setWeb3(null);
    setContract(null);
    setStoredValue(null);
    toast.info("Wallet disconnected.");
  };

  const fetchStoredValue = async (contractRef = contract) => {
    try {
      if (contractRef) {
        const value = await contractRef.methods.get().call();
        setStoredValue(value.toString());
      }
    } catch (err) {
      toast.error("Failed to fetch data.");
      console.error(err);
    }
  };

  const handleSet = async () => {
    if (contract && inputValue && web3 && walletAddress) {
      try {
        setLoading(true);
        toast.info("Transaction submitted...");
        await contract.methods.set(inputValue).send({ from: walletAddress });
        setInputValue('');
        toast.success("Value updated successfully!");
        fetchStoredValue();
      } catch (err) {
        toast.error("Transaction failed.");
```


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

* Implementation Phase: Final Output (no error)

Now your wallet is successfully connected with your DApp



* Observations

1. The lab demonstrates how to integrate a blockchain smart contract with a frontend application using Web3.js, enabling real-time interaction between users and the blockchain.
2. It highlights connecting wallets, reading/writing contract data, and handling blockchain events from the UI.

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.*