



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 Experiment :** React Start – DApp Frontend Scaffolding

### \* Coding Phase: Pseudo Code / Flow Chart / Algorithm

#### ALGORITHM:

1. Start
2. Open remix IDE write the smart contract in SimpleStorage.sol
3. Compile the smart contract in remix
4. Copy the generated abi and save it somewhere
5. Deploy the contract in sepolia testnet using metamask
6. Copy the deployed contract address
7. Create a react frontend project using create react app
8. Add the contract address and network information in .env file
9. Install web3.js
10. Use the ABI and contract address to connect the frontend with smart contract
11. End

### \* Software used

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

## \* 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;
    }
}
```

### ABI key


```
export const simpleStorageABI =
[
  {
    "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": [
      {
        "internalType": "uint256",
        "name": "x",
        "type": "uint256"
      }
    ],
    "name": "set",
    "outputs": [],
    "stateMutability": "nonpayable",
```

```
    "type": "function"
  },
  {
    "inputs": [],
    "name": "storedData",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  }
]
```

## \* Implementation Phase: Final Output (no error)




After compilation deploy the smart contract in sepolia test network using metamask

**DEPLOY & RUN TRANSACTIONS**

ENVIRONMENT 

Injected Provider - MetaMask

Sepolia (11155111) network

ACCOUNT   

0x5b3...5c960 (0.4295216765...)

**+ Create Smart Account**

GAS LIMIT

☒ Estimated Gas

☐ Custom 3000000

VALUE

0 Wei

CONTRACT

SimpleStorage - SimpleStorage.sol

evm version: prague

**Deploy** 123

☐ Publish to IPFS


**At Address** Load contract from Address


MetaMask


Account 1  
Sepolia


**Deploy a contract**

This site wants you to deploy a contract

Estimated changes  No changes

Request from  remix.ethereum.org

Network fee  0.0003 s SepoliaETH

Speed  Market ~12 sec

**Cancel** **Confirm**

Type the library name to see available commands.  
creation of SimpleStorage pending...

[view on Etherscan](#) [view on Blockscout](#)

✓ [block:8976963 txIndex:31] from: 0x5b3...5c960 to: SimpleStorage.(constructor) value: 0 wei data: 0x608...0007b logs: 0 hash: 0xae9...09e10

**Debug** 

## \* 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.

```

+ .env U X
frontend > + .env
1 REACT_APP_CONTRACT_ADDRESS=0xf00A07027fF338c8e8fdb29f094B5Ff8Fc0e6831
2 REACT_APP_NETWORK=sepolia

```

```

App.js U X
frontend > src > App.js > default
1 import React, { useEffect, useState } from 'react';
2 import Web3 from 'web3';
3 import { SimpleStorageABI } from './abi';
4 import { ToastContainer, toast } from 'react-toastify';
5 import 'react-toastify/dist/ReactToastify.css';
6 import { FaSpinner } from 'react-icons/fa';
7
8 const contractAddress = process.env.REACT_APP_CONTRACT_ADDRESS;
9
10 function App() {
11   const [walletAddress, setWalletAddress] = useState(null);
12   const [web3, setWeb3] = useState(null);
13   const [contract, setContract] = useState(null);
14   const [storedValue, setStoredValue] = useState(null);
15   const [inputValue, setInputValue] = useState('');
16   const [loading, setLoading] = useState(false);
17
18   const connectWallet = async () => {
19     if (window.ethereum) {
20       try {
21         const web3Instance = new Web3(window.ethereum);
22         await window.ethereum.request({ method: 'eth_requestAccounts' });
23         const accounts = await web3Instance.eth.getAccounts();
24
25         const contractInstance = new web3Instance.eth.Contract(SimpleStorageABI, contractAddress);
26
27         setWalletAddress(accounts[0]);
28         setWeb3(web3Instance);
29         setContract(contractInstance);
30
31         toast.success("Wallet connected!");
32         fetchStoredValue(contractInstance);
33       } catch (err) {
34         toast.error("Connection failed.");
35         console.error(err);
36       }
37     } else {
38       toast.error("Please install MetaMask.");
39     }
40   };
41
42   const disconnectWallet = () => {
43     setWalletAddress(null);
44     setWeb3(null);
45     setContract(null);
46     setStoredValue(null);
47     toast.info("Wallet disconnected.");
48   };
49
50   const fetchStoredValue = async (contractRef = contract) => {
51     try {
52       if (contractRef) {
53         const value = await contractRef.methods.get().call();
54         setStoredValue(value.toString());
55       }
56     } catch (err) {
57       toast.error("Failed to fetch data.");
58       console.error(err);
59     }
60   };
61
62   const handleSet = async () => {
63     if (contract && inputValue && web3 && walletAddress) {
64       try {
65         setLoading(true);
66         toast.info("Transaction submitted...");
67         await contract.methods.set(inputValue).send({ from: walletAddress });
68         setInputValue('');
69         toast.success("Value updated successfully!");
70         fetchStoredValue();
71       } catch (err) {
72         toast.error("Transaction failed.");
73       }
74     } finally {
75       setLoading(false);
76     }
77   };
78
79   return (
80     <div style={{
81       padding: '30px',
82       fontFamily: 'Segoe UI, sans-serif',
83       background: 'linear-gradient(to right, #0f2027, #203a43, #2c5364)',
84       color: 'white',
85       minHeight: '100vh'
86     }}>
87       <ToastContainer />
88       <div style={{
89         maxWidth: '500px',
90         margin: 'auto',
91         background: '#1e2a38',
92         padding: '30px',
93         borderRadius: '15px',
94         boxShadow: '0 10px 20px rgba(0,0,0,0.3)'
95       }}>
96         <h1 style={{
97           textAlign: 'center',
98           marginBottom: '20px',
99           color: '#61dafb'
100         }}> Simple Storage DApp</h1>
101
102         {walletAddress ? (
103           <button
104             onClick={connectWallet}
105             style={buttonStyle}>
106             Connect MetaMask Wallet

```

```

108 </button>
109 ) : (
110   <div>
111     <p>strong>Connected:</strong> <span style={{ color: '#80e9fd' }}>{walletAddress}</span></p>
112     <button onClick={disconnectWallet} style={{ ...buttonStyle, backgroundColor: '#ff5555', marginBottom: '20px' }}>
113       Disconnect Wallet
114     </button>
115
116     <p>strong>Stored Value:</strong> <span style={{ color: '#50fa7b' }}>{storedValue}</span></p>
117
118     <div style={{ marginTop: '10px' }}>
119       <input
120         type="number"
121         placeholder="Enter new value"
122         value={inputValue}
123         onChange={(e) => setInputValue(e.target.value)}
124         style={inputStyle}
125       />
126       <button
127         onClick={handleSet}
128         disabled={loading}
129         style={{ ...buttonStyle, marginLeft: '10px' }}>
130         <div>
131           {loading ? <FaSpinner className="spin" /> : 'Update'}
132         </div>
133       </button>
134
135       <button
136         onClick={() => fetchStoredValue()}
137         style={{ ...buttonStyle, backgroundColor: '#622da4', marginTop: '20px' }}>
138         <div>
139           Retrieve Latest Data
140         </div>
141       </button>
142     </div>
143   </div>

```

## \* Implementation Phase: Final Output (no error)

Now open terminal and run the project by writing the code `npm start`

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Compiled successfully!

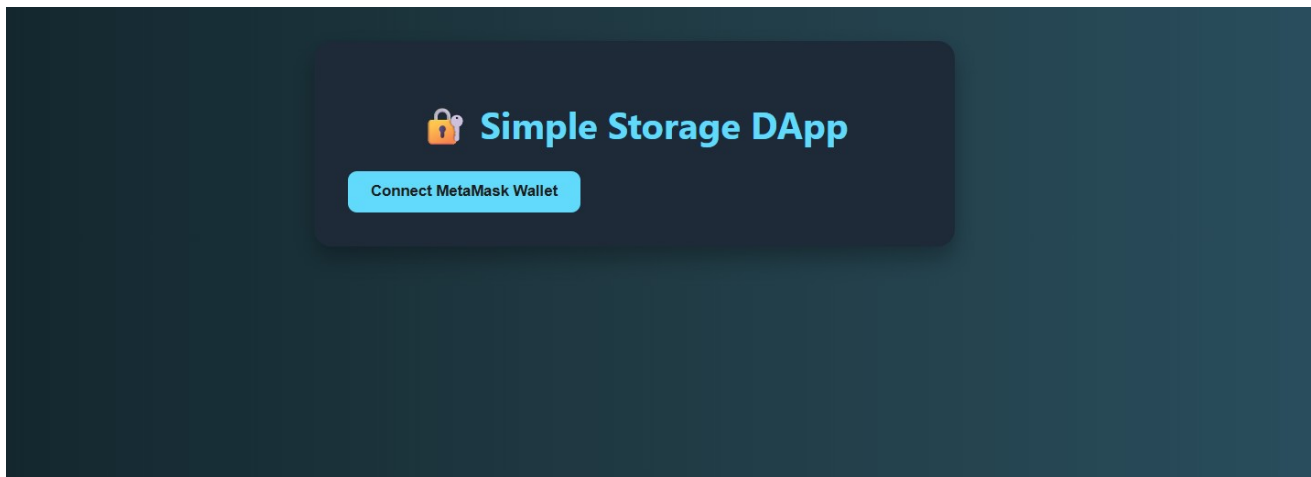
You can now view frontend in the browser.

Local:      http://localhost:3000
On Your Network: http://10.99.38.54:3000

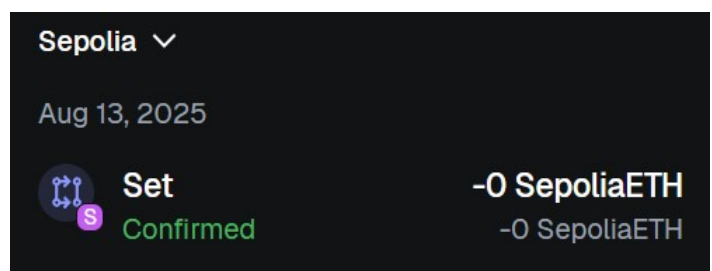
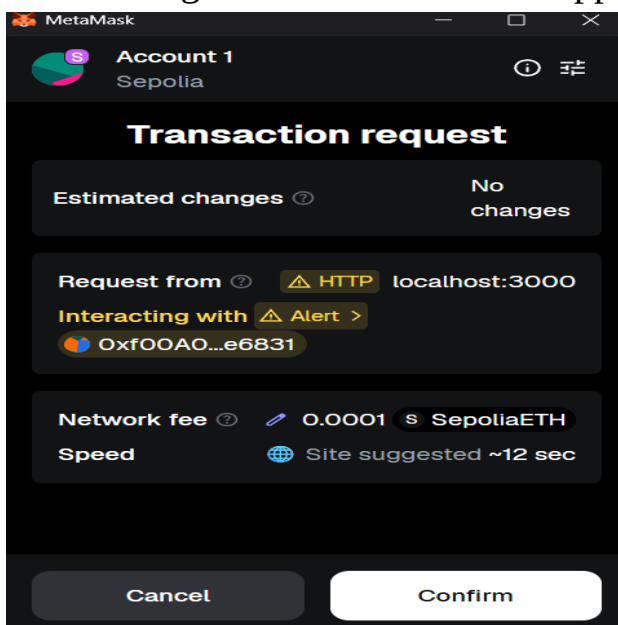
Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully

```



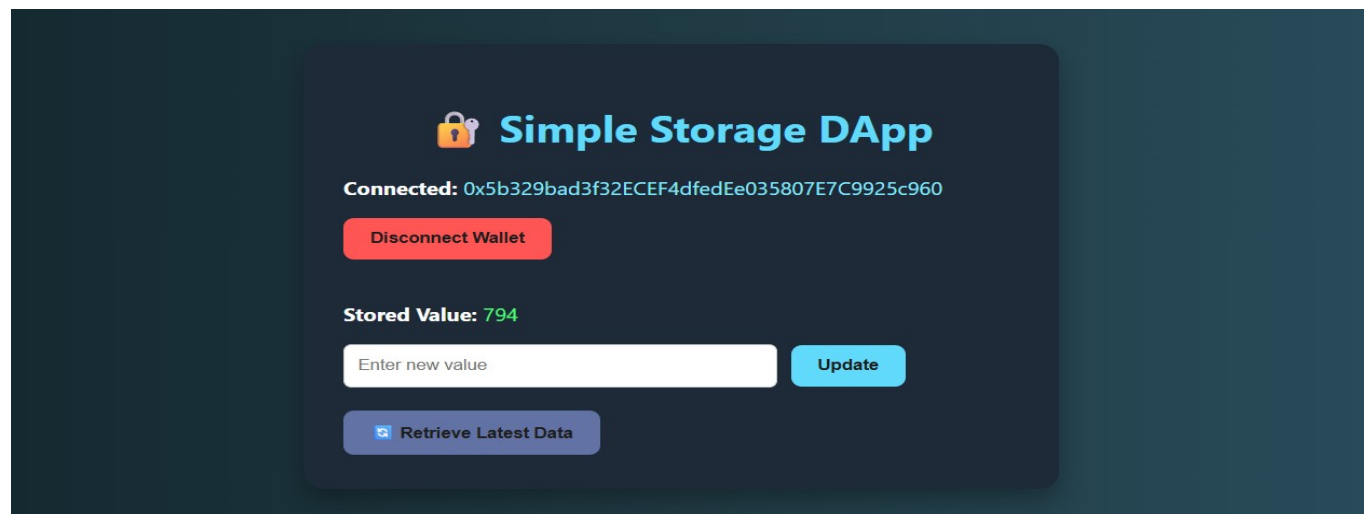
Connecting the wallet with the DApp



## \* Implementation Phase: Final Output (no error)

Applied and Action Learning

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		
<b>Total</b>	<b>50</b>		

**Signature of the Student:**

**Name :**

**Regn. No. :**

**Signature of the Faculty:**

Page No.....

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