



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 :** Connect the dots – Ether.js find MetaMask UI

### \* 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

## \* Testing Phase: Compilation of Code (error detection)

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

```

1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3  contract Counter {
4      uint public count;
5      constructor(uint _start) {  Infinite gas 132800 gas
6          count = _start;
7      }
8      function increment() public {  Infinite gas
9          count += 1;
10     }
11     function decrement() public {  Infinite gas
12         require(count > 0, "Counter is already at zero");
13         count -= 1;
14     }
15     function getCount() public view returns (uint) {  2453 gas
16         return count;
17     }
18 }

```

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

```

1  [
2  {
3      "inputs": [
4          {
5              "internalType": "uint256",
6              "name": "_start",
7              "type": "uint256"
8          }
9      ],
10     "stateMutability": "nonpayable",
11     "type": "constructor"
12 },
13 {
14     "inputs": [],
15     "name": "count",
16     "outputs": [
17         {
18             "internalType": "uint256",
19             "name": "",
20             "type": "uint256"
21         }
22     ],
23     "stateMutability": "view",
24     "type": "function"
25 },
26 ]

```

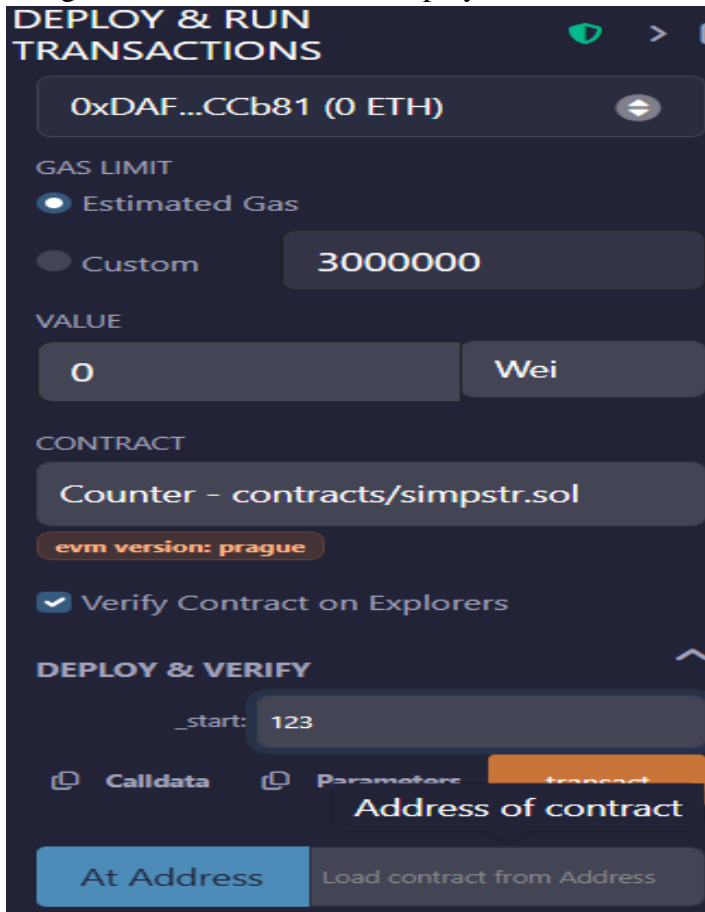
```

26  {
27      "inputs": [],
28      "name": "decrement",
29      "outputs": [],
30      "stateMutability": "nonpayable",
31      "type": "function"
32 },
33  {
34      "inputs": [],
35      "name": "getCount",
36      "outputs": [
37          {
38              "internalType": "uint256",
39              "name": "",
40              "type": "uint256"
41          }
42      ],
43      "stateMutability": "view",
44      "type": "function"
45 },
46  {
47      "inputs": [],
48      "name": "increment",
49      "outputs": [],
50      "stateMutability": "nonpayable",
51      "type": "function"
52 },
53 ]

```

## \* 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

## \* 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 store contract address and testnet network

```
C: > Users > amik > OneDrive > Desktop > SIMPLESTORAGE_FRONTEND > addfile.js > lab4 ipfs > .env
1 1.REACT_APP_CONTRACT_ADDRESS=0xdafe4dfdda3e39e0580d206bcfb59b93caecb81
2 2.REACT_APP_NETWORK=sepolia
```

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

```
1 import React, { useEffect, useState } from "react";
2 import { ethers } from "ethers";
3 import { CONTRACT_ABI } from "./abi";
4
5 // Replace with your deployed contract address from Remix
6 const CONTRACT_ADDRESS = "0xa62463A56EE90742F818920F56cBc48696e80a";
7
8 function App() {
9   const [provider, setProvider] = useState(null);
10   const [signer, setSigner] = useState(null);
11   const [account, setAccount] = useState("");
12   const [contract, setContract] = useState(null);
13   const [inputValue, setInputValue] = useState("");
14   const [storedValue, setStoredValue] = useState("");
15   const [loading, setLoading] = useState(false);
16   const [status, setStatus] = useState("");
17
18   // Connect Wallet (ethers.js)
19   const connectWallet = async () => {
20     if (!window.ethereum) {
21       alert("Please install MetaMask to use this App.");
22       return;
23     }
24
25     try {
26       const ethProvider = new ethers.providers.Web3Provider(window.ethereum, "any");
27       // prompt user for account connections
28       await ethProvider.send("eth_requestAccounts", []);
29       const ethSigner = ethProvider.getSigner();
30       const address = await ethSigner.getAddress();
31
32       setProvider(ethProvider);
33       setSigner(ethSigner);
34       setAccount(address);
35
36       const contractInstance = new ethers.Contract(CONTRACT_ADDRESS, CONTRACT_ABI, ethSigner);
37       setContract(contractInstance);
38
39       // read stored value (may be 0 or number)
40       const currentValue = await contractInstance.get();
41       setStoredValue(currentValue.toString());
42
43       setStatus("🟢 Wallet connected");
44     } catch (error) {
45       console.error("Wallet connection error:", error);
46       setStatus("❌ failed to connect wallet.");
47     }
48   };
49
50   // Handle Submit (ethers.js)
51   const handleSubmit = async () => {
52     if (!inputValue) return alert("Please enter a number.");
53     if (!contract || !signer || !account) return alert("Wallet not connected.");
54
55     try {
56       setLoading(true);
57       setStatus("🔄 Sending transaction...");
58
59       // contract.set returns a TransactionResponse
60       const tx = await contract.set(inputValue);
61       // wait for transaction to be mined
62       await tx.wait();
63
64       const updatedValue = await contract.get();
65       setStoredValue(updatedValue.toString());
66       setInputValue("");
67       setStatus("🟢 Number updated on blockchain!");
68     } catch (error) {
69       console.error(error);
70       setStatus("❌ Transaction failed.");
71     } finally {
72       setLoading(false);
73     }
74   };
75
76   // optional: auto-connect if already authorized (safe to call once)
77   useEffect(() => {
78     // detect if MetaMask already connected
79     const checkConnected = async () => {
80       if (window.ethereum) {
81         try {
82           const ethProvider = new ethers.providers.Web3Provider(window.ethereum, "any");
83           const accounts = await ethProvider.listAccounts();
84           if (accounts.length > 0) {
85             // set up signer and contract like in connectWallet
86             const ethSigner = ethProvider.getSigner();
87             const address = accounts[0];
88             setProvider(ethProvider);
89             setSigner(ethSigner);
90             setAccount(address);
91
92             const contractInstance = new ethers.Contract(CONTRACT_ADDRESS, CONTRACT_ABI, ethSigner);
93             setContract(contractInstance);
94             const currentValue = await contractInstance.get();
95             setStoredValue(currentValue.toString());
96             setStatus("🟢 wallet connected");
97           }
98         } catch (err) {
99           // ignore
100         }
101       }
102     };
103     checkConnected();
104   }, []);
```

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

```

106   return (
107     <div style={{ padding: "40px", fontFamily: "Arial", maxWidth: "600px", margin: "auto" }}>
108       <h2> Simple Storage DApp (ethers.js)</h2>
109
110       {account ? (
111         <button
112           onClick={connectWallet}
113           style={{
114             padding: "10px 20px",
115             backgroundColor: "#007bff",
116             color: "white",
117             border: "none",
118             borderRadius: "5px",
119             marginBottom: "20px",
120           }}
121         >
122           Connect Wallet
123         </button>
124       ) : (
125         <div style={{ marginBottom: "20px" }}>
126           <p><strong>🔴 Wallet:</strong> {account}</p>
127           <p><strong>🟡 Stored Number:</strong> {storedValue}</p>
128         </div>
129       )}
130
131       {account && (
132         <div>
133           <input
134             type="number"
135             value={inputValue}
136             onChange={(e) => setInputValue(e.target.value)}
137             placeholder="Enter a number"
138             style={{ padding: "8px", width: "70%", marginRight: "5px" }}
139             disabled={loading}
140           />
141           <button
142             onClick={handleSubmit}
143             disabled={loading}
144             > {loading ? "🔄 Updating..." : "Set Number"}
145           </button>
146         </div>
147
148         {status && (
149           <p style={{ marginTop: "20px", color: status.includes("✅") ? "green" : "red" }}>
150             {status}
151           </p>
152         )}
153       )}
154     </div>
155   );
156 }
157
158 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
```

## \* 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		
<b>Total</b>	<b>50</b>		

**Signature of the Student:**

Name :

Regn. No. :

**Signature of the Faculty:**

Page No.....  
Page No.....

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