



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 – Ethers.js and MetaMask UI

*** Coding Phase: Pseudo Code / Flow Chart /**

Algorithm

- |
- | Compile the smart contract using the **Solidity compiler** to generate the ABI.
- | Deploy the contract on **Sepolia Testnet** using **MetaMask (Injected Provider)**.
- | Copy the **deployed contract address**.
- | Create a new **React project** using create-react-app.
- | Inside the src folder, create an ABI.js file and paste the contract ABI.
- | In the project root, create a **.env file** to securely store the contract address and network details.
- | Install required dependencies (ethers.js as primary library, web3.js only if needed).
- | In App.js, implement wallet connection and blockchain interaction logic using **ethers.js**.
- | Build a simple **UI** to store and retrieve values from the contract.
- | Run the project using npm start and test interactions via MetaMask popups.

*** Software used**

1. MetaMask Wallet
2. Remix IDE.
3. MS Word.
4. Brave for researching.

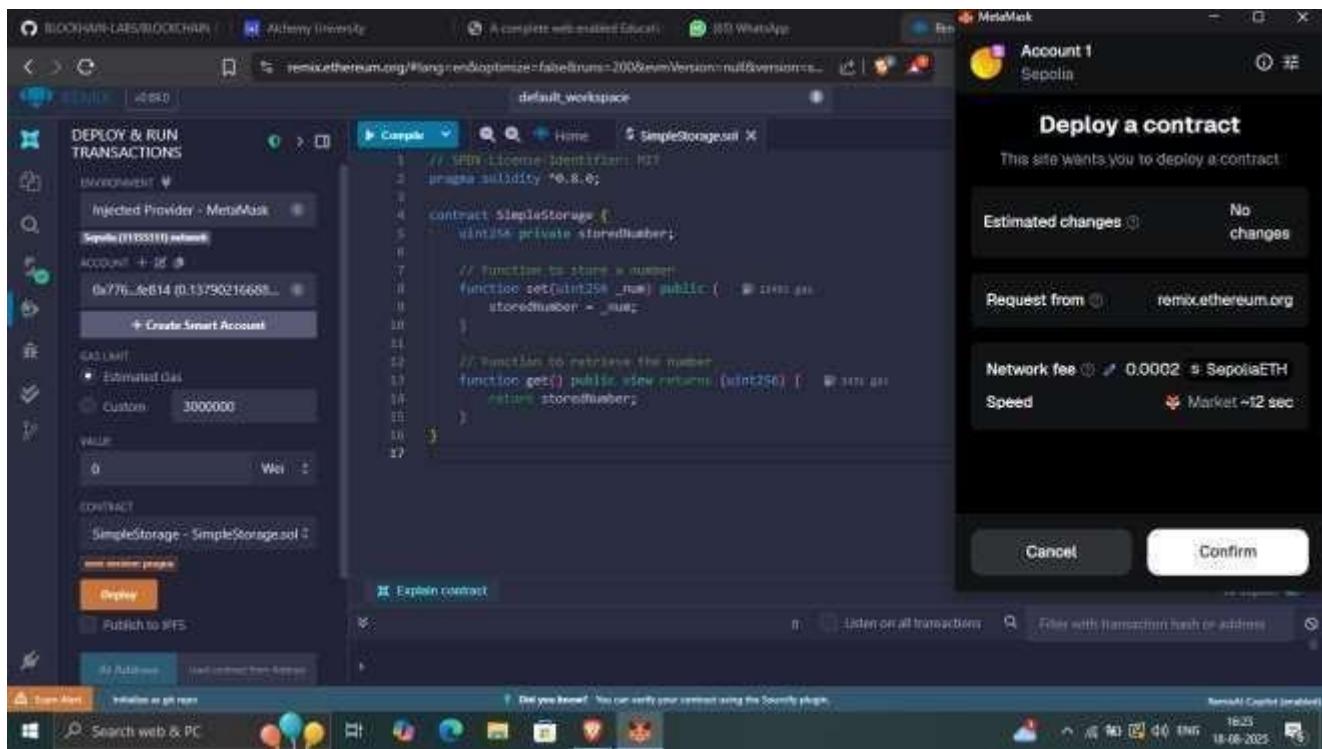
* Implementation Phase: Final Output (no error)

1. Firstly go to remix ide and write a smart contract on simplestorage.sol and compile it.
2. After compilation ABI will generated.
3. Then go to deploy and run transactions section and choose environment as injected provider-metamask, then simply deploy it.
4. 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.
5. Now in app.js write the frontend code and wallet connection code also.
6. Now simply move forward terminal of V.S Code just run the project with command npm start.

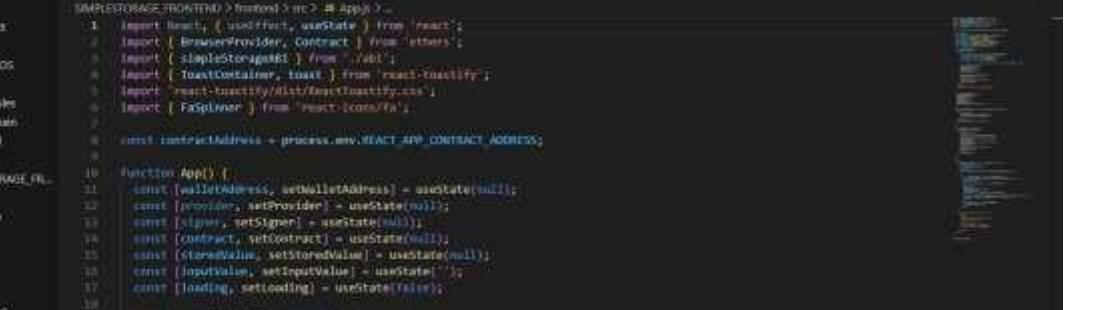
```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract SimpleStorage {
5     uint256 private storedNumber;
6
7     // Function to store a number
8     function set(uint256 _num) public {
9         storedNumber = _num;
10    }
11
12     // Function to retrieve the number
13     function get() public view returns (uint256) {
14         return storedNumber;
15    }
16 }
17

```



* Implementation Phase: Final Output (no error)



The screenshot shows a Microsoft Edge browser window with a React application running at localhost:3001. The application has a header "React App" and a main form area with three input fields: "Name", "Age", and "Email". Below the form is a "Submit" button. The browser's developer tools are open, showing the DOM tree and the Network tab.

```
import React, { useState, useEffect } from 'react';
import { BrowserRouter, Contract } from '@ethers';
import { SimpleStorageABI } from './abi';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import { Fingerprint } from 'react-fingerprint';

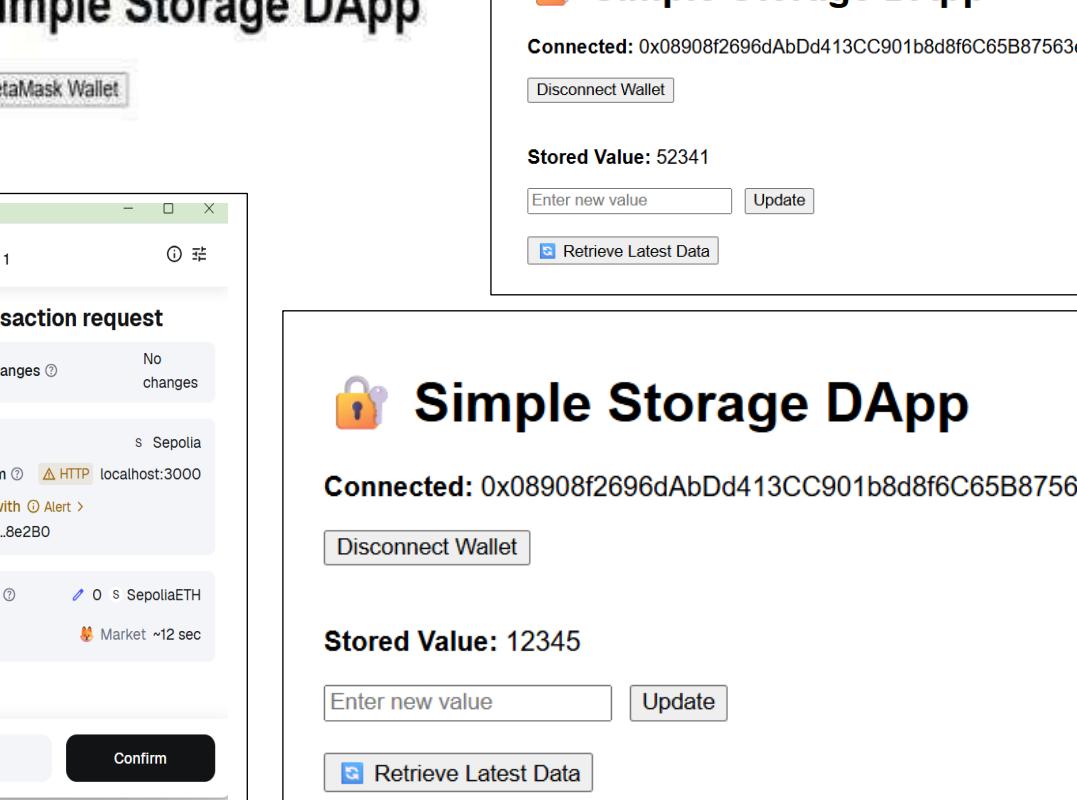
const contractAddress = process.env.REACT_APP_CONTRACT_ADDRESS;

function App() {
  const [walletAddress, setWalletAddress] = useState(null);
  const [provider, setProvider] = useState(null);
  const [signer, setSigner] = useState(null);
  const [contract, setContract] = useState(null);
  const [storeValue, setStoreValue] = useState(null);
  const [inputValue, setInputValue] = useState('');
  const [loading, setLoading] = useState(false);

  const connectWallet = async () => {
    if (window.ethereum) {
      try {
        const provider = new BrowserProvider(window.ethereum);
        setProvider(provider);
        const signer = provider.getSigner();
        setSigner(signer);
        const contract = await SimpleStorageABI.attach(
          contractAddress,
          signer
        );
        setContract(contract);
        const storeValue = await contract.store('Hello');
        setStoreValue(storeValue);
      } catch (err) {
        console.error(err);
      }
    }
  };

  return (
    <div>
      <h1>React App</h1>
      <div>
        <input type="text" value={inputValue} onChange={e => setInputValue(e.target.value)} />
        <button onClick={connectWallet}>Connect Wallet</button>
        <button onClick={() => setStoreValue('Hello')}>Store Value</button>
        <button onClick={() => toast.info(`Value stored: ${storeValue}`)}>Get Value</button>
      </div>
      <pre>{JSON.stringify(storeValue, null, 2)}</pre>
    </div>
  );
}

export default App;
```



The image shows three views of a DApp interface. The top-left view is the main DApp page with a yellow padlock icon, the title "Simple Storage DApp", a "Connect MetaMask Wallet" button, and a "Connected" status bar showing a wallet address. The top-right view shows the "Stored Value" as 52341 with an "Enter new value" input field and an "Update" button. The bottom view shows a "Transaction request" dialog from MetaMask, listing details like "Estimated changes: No changes", "Network: Sepolia", "Request from: localhost:3000", "Interacting with: Alert", and "Network fee: 0 SepoliaETH". It includes "Cancel" and "Confirm" buttons. The bottom-right view is identical to the top-right view, showing the stored value and interaction buttons.

Simple Storage DApp

Connected: 0x08908f2696dAbDd413CC901b8d8f6C65B87563e1

Disconnect Wallet

Stored Value: 52341

Enter new value Update

Retrieve Latest Data

Simple Storage DApp

Connected: 0x08908f2696dAbDd413CC901b8d8f6C65B87563e1

Disconnect Wallet

Stored Value: 12345

Enter new value Update

Retrieve Latest Data

* Observations:

- | Smart contract successfully deployed on **Sepolia Testnet**.
- | **MetaMask wallet connection** worked with the React frontend.
- | UI allowed users to **store and retrieve values**.
- | **ethers.js** ensured a secure and modern interaction with the contract.
- | **.env** file maintained contract details securely.
- | The setup provides a **scaffold for future DApp development**

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 :

Signature of the Faculty:

Regn. No. :