**Unit 4 – Blockchain and Decentralized Applications**

**Lab 1 Manual**

**Part A: Introduction to Solidity Programming with Remix IDE**

**Objective:**

By the end of this lab, students will:

* Understand the basics of Solidity.
* Write a simple smart contract.
* Deploy and interact with the contract using Remix IDE.

**Step 1:** Setting Up Remix IDE ([Remix IDE](about:blank)) a web-based IDE requires no installation

**Step 2:** Write a Simple Smart Contract i.e., Simple "Hello Ethereum" Contract

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0; // compiler version

contract HelloEthereum {

string public message; // message is a public state variable storing a string

// Constructor that initializes the contract with a message

constructor(string memory initialMessage) { //The contract’s constructor initializes a message with a value when the contract is deployed.

message = initialMessage;

}

// Function to update the message

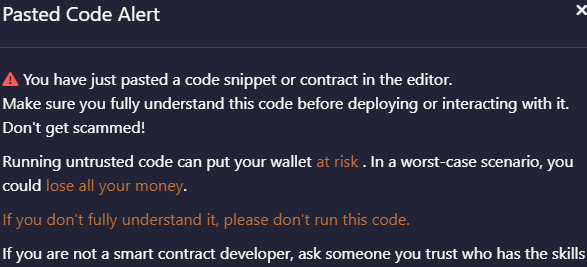
function updateMessage(string memory newMessage) public { // updateMessage is a function allowing anyone to change the stored message.

message = newMessage;

}

}

**Step 3:** Compile the Smart ContractHelloEthereum.sol (see a green checkmark once compiled). Please beware:



**Step 4:** Navigate to the "Deploy & Run Transactions" tab and choose ‘JavaScript VM()’ to simulate the EVM. Under ‘Deploy', enter a string for the initialMessage (e.g., "Hello, Ethereum!") then click "Deploy."

Once deployed, you will see the contract under the "Deployed Contracts" section.

**Step 5:** Read the message variable by clicking the message button to view the current message stored in the contract. Update the message by using the updateMessage function, input a new message (e.g., "Blockchain is cool!"), and submit the transaction. Check the updated message by clicking the message button again.

**Step 6:** Every interaction with the contract costs gas. The gas used in deploying the contract and updating the message can be seen in the "Transaction Logs." Any unused gas is refunded. You can see how a higher gas limit is specified than required and how the remaining gas is returned.

**Challenge: Add Another Function**

Add a new function ‘getMessage’ that returns the current message. This function should be viewable by anyone without changing the state (Hint: use view keyword).

Solution:

function getMessage() public view returns (string memory) {

return message;

}

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

**Part B Case 1: Task on Solidity Programming with VS Code, Metamask, OpenZeppelin SDK, Node.js and Sepolia Testnet**

**Objective:**

By the end of this lab, students will:

* Set up Visual Studio Code Editor ([VS Code](https://code.visualstudio.com)) for Solidity development using OpenZeppelin SDK.
* Install Metamask for interacting with the Sepolia testnet.
* Write, compile, and deploy a Solidity smart contract using the OpenZeppelin SDK.

In **case 1**, we use **Node.js** because it is a common environment for working with tools like **OpenZeppelin SDK**, and similar frameworks, which are built around JavaScript and run on Node.js. Node.js is used for managing packages, running scripts, and compiling Solidity contracts. This approach is well-documented and widely supported.

**Step 1:** Download and Install VS Code from based on your operating system. Also, download and install the LTS version of [Node.js](https://nodejs.org/). Verify installation by running the two commands (i. node -v and ii. npm -v) in the terminal:

**Step 2:** Open VS Code and Install Solidity Extension by clicking on the Extensions tab (or Ctrl+Shift+X), and search for "Solidity" by Juan Blanco. Run the following command to install OpenZeppelin contracts library:

npm install @openzeppelin/contracts

**Step 3:** Install and Set Up Metamask **(**[Metamask](https://metamask.io/)) and install the browser extension (available for Chrome, Firefox).Follow the instructions to create your wallet. Backup your recovery phrase in a safe place.

Once the Metamask is open, click on the network dropdown at the top, and select "Sepolia Test Network". If Sepolia is not visible (don’t panic), enable it by going to **Settings > Networks** and turning on the testnets option.

Visit a faucet like [Sepolia Faucet](https://sepoliafaucet.com/) and enter your wallet address from Metamask to receive testnet ETH. This testnet ETH will be used to deploy contracts without spending real money.

**Step 4:** Install OpenZeppelin CLI by opening the terminal inside VS Code and install the OpenZeppelin CLI globally by running the following command:

npm install -g @openzeppelin/cli

In addition,in the terminal, navigate to a new project folder, then run the following commands to initialize your OpenZeppelin project:

mkdir my-solidity-project

cd my-solidity-project

npx openzeppelin init

**Step 5:** Copy the same simple Solidity Smart Contract which you run in Remix IDE

**Step 6:** Configure deployment to Sepolia Testnet by adding Sepolia network configuration to deploy the contract to the Sepolia testnet, you must configure your network settings in the OpenZeppelin configuration file. You can do this by creating a .openzeppelin folder and inside it, add a networks.js file:

module.exports = {

networks: {

sepolia: {

provider: () =>

new HDWalletProvider(

'YOUR\_METAMASK\_PRIVATE\_KEY', // Metamask private key (never share this publicly)

'https://eth-sepolia.g.alchemy.com/v2/YOUR \_API\_KEY'

),

network\_id: 11155111, // Sepolia Network ID

gasPrice: 10000000000, // 10 gwei

},

},

};

**Step 7:** Before proceeding ensure there are no errors Compile the Smart Contract by running the following command in your terminal to compile the Solidity code using the OpenZeppelin CLI:

npx openzeppelin compile

**Step 8:** Deploy the Contract to Sepolia Testnet by running the following command to deploy your smart contract to the Sepolia testnet but wait for the deployment to complete. The terminal will output the address of your deployed contract:

npx openzeppelin deploy --network sepolia

**Step 9:** Open and use the OpenZeppelin CLI after deploying the contract to interact (read, update and verify) with the deployed contract.In the terminal, run the following to read and interact with your deployed contract:

npx oz call --network sepolia --contract HelloWorld --method message

Run the following to update the contract’s state (message) using the contract function of deployed contract:

npx oz send-tx --network sepolia --contract HelloWorld --method updateMessage --args "New Message!"

Run the following to verify the updated message of deployed contract:

npx oz call --network sepolia --contract HelloWorld --method message

**Heads up:**

* **Gas Fee Consideration**: Deploying and interacting with contracts on Ethereum (even on testnets) incurs gas fees in the form of testnet ETH. Repeatedly compiling and redeploying contracts costs more in gas fees.
* **Testnet Use**: Testnet like Sepolia is great for learning and experimenting without spending real ETH, but testnet ETH can be limited, so avoid unnecessary redeployments.
* **Debugging**: Thoroughly test your code using the OpenZeppelin CLI before deploying to a live network.

**OR**

**Part B Case 2: Task on Solidity Programming with VS Code, Metamask, Brownie, and Sepolia Testnet**

**Objective:**

By the end of this lab, students will:

* Set up Visual Studio Code for Solidity development using Python and Brownie.
* Install Metamask for interacting with the Sepolia testnet.
* Write, compile, and deploy a Solidity smart contract using Brownie.

**Using Python Instead of Node.js**

In case 2, if you want to use Python instead, you can leverage Brownie, a Python-based framework for Ethereum smart contract development. Brownie integrates smoothly with Python, making it a suitable alternative for developers who prefer Python over JavaScript.

**Step 1:** Download and Install VS Code from based on your operating system.

**Step 2:** Install latest Python ([Python](https://www.python.org/downloads/)) and verify installation by running (python --version) in the terminal. Also, install Brownie by running (i. pip install eth-brownie and ii. brownie --version) in terminal

**Step 3:** Install and Set Up Metamask **(**[Metamask](https://metamask.io/)) and install the browser extension (available for Chrome, Firefox).Follow the instructions to create your wallet. Backup your recovery phrase in a safe place.

**Step 4:** Set Up Brownie Project by opening VS Code terminal and create new directory for project

**mkdir brownie\_solidity\_project**

**cd brownie\_solidity\_project**

**brownie init**

**Step 5:** Compile the Smart Contract by running (brownie compile) in terminal. Check for errors and ensure your contract compiles successfully.

**Step 6:** Configure Brownie to Use Sepolia Testnet by adding Sepolia Network Configuration as discussed above

**Step 7:** Set Up Your Wallet by exporting your private key from Metamask and set it in Brownie by running (export PRIVATE\_KEY="YOUR\_METAMASK\_PRIVATE\_KEY")

**Step 8:** Deploy the contract to Sepolia testnet by creating a new Python script inside the scripts folder (deploy\_hello.py) with an initial message, "Hello, Blockchain!"

from brownie import HelloWorld, accounts

def main():

acct = accounts.load("deployer\_account")

HelloWorld.deploy("Hello, Blockchain!", {'from': acct})

Run the following command in the terminal to deploy the contract:

brownie run scripts/deploy\_hello.py --network sepolia

**Step 9:** Deploy the contract by noting the contract address from the terminal output and interact with the contract using Brownie’s CLI to call the message function and check the initial message:

brownie console --network sepolia

HelloWorld[-1].message()

Update the message by calling the updateMessage function to update the message:

HelloWorld[-1].updateMessage("New Message!", {'from': accounts[0]})

Verify that the message has been updated:

HelloWorld[-1].message()