!pip install web3

!pip install "jsonschema<4.0"

from web3 import Web3

!pip3 install py-solc-x

from solcx import compile\_standard, install\_solc

from solcx import compile\_source

from web3 import EthereumTesterProvider

from solcx import compile\_standard, install\_solc

install\_solc("0.8.0")

from solcx import compile\_source

from web3 import Web3

Web3

from web3 import Web3

# Connect to Ethereum node (using Infura in this example)

provider\_url='https://sepolia.infura.io/v3/c4340ff166d442efa85c23268ad85bb0'

w3=Web3(Web3.HTTPProvider(provider\_url))

w3.is\_connected()

# Check if the connection is successful

if not w3.isConnected():

    raise Exception("Failed to connect to the Ethereum network")

# Smart contract ABI (Application Binary Interface)

contract\_abi = '''[

  {

    "anonymous": false,

    "inputs": [

      {

        "indexed": true,

        "internalType": "address",

        "name": "meterAddress",

        "type": "address"

      },

      {

        "indexed": false,

        "internalType": "uint256",

        "name": "timestamp",

        "type": "uint256"

      },

      {

        "indexed": false,

        "internalType": "string",

        "name": "hexData",

        "type": "string"

      },

      {

        "indexed": false,

        "internalType": "string",

        "name": "latitude",

        "type": "string"

      },

      {

        "indexed": false,

        "internalType": "string",

        "name": "longitude",

        "type": "string"

      }

    ],

    "name": "NewMeterData",

    "type": "event"

  },

  {

    "inputs": [

      {

        "internalType": "address",

        "name": "\_meterAddress",

        "type": "address"

      }

    ],

    "name": "getMeterData",

    "outputs": [

      {

        "internalType": "uint256",

        "name": "timestamp",

        "type": "uint256"

      },

      {

        "internalType": "string",

        "name": "hexData",

        "type": "string"

      },

      {

        "internalType": "string",

        "name": "latitude",

        "type": "string"

      },

      {

        "internalType": "string",

        "name": "longitude",

        "type": "string"

      }

    ],

    "stateMutability": "view",

    "type": "function"

  },

  {

    "inputs": [

      {

        "internalType": "address",

        "name": "",

        "type": "address"

      }

    ],

    "name": "meterData",

    "outputs": [

      {

        "internalType": "uint256",

        "name": "timestamp",

        "type": "uint256"

      },

      {

        "internalType": "string",

        "name": "hexData",

        "type": "string"

      }

    ],

    "stateMutability": "view",

    "type": "function"

  },

  {

    "inputs": [

      {

        "internalType": "address",

        "name": "",

        "type": "address"

      }

    ],

    "name": "meterLocations",

    "outputs": [

      {

        "internalType": "string",

        "name": "latitude",

        "type": "string"

      },

      {

        "internalType": "string",

        "name": "longitude",

        "type": "string"

      }

    ],

    "stateMutability": "view",

    "type": "function"

  },

  {

    "inputs": [

      {

        "internalType": "string",

        "name": "\_password",

        "type": "string"

      },

      {

        "internalType": "string",

        "name": "\_hexData",

        "type": "string"

      },

      {

        "internalType": "string",

        "name": "\_latitude",

        "type": "string"

      },

      {

        "internalType": "string",

        "name": "\_longitude",

        "type": "string"

      }

    ],

    "name": "recordMeterData",

    "outputs": [],

    "stateMutability": "nonpayable",

    "type": "function"

  },

  {

    "inputs": [

      {

        "internalType": "string",

        "name": "\_password",

        "type": "string"

      }

    ],

    "name": "setPassword",

    "outputs": [],

    "stateMutability": "nonpayable",

    "type": "function"

  }

]'''

# Replace with your actual contract address and wallet information

contract\_address = Web3.toChecksumAddress('0xc862225BccF703EB298AD1158334ea69299D4886')

wallet = Web3.toChecksumAddress('0xc862225BccF703EB298AD1158334ea69299D4886')

private\_key = 'd08f96c9f5b823d4dc3e24e1feaf8b3211d1afdd87b3df211e1c57ae745fea55'

# Create a contract instance

contract\_instance = w3.eth.contract(address=contract\_address, abi=contract\_abi)

# Parameters for the recordMeterData function

password = "YourSecurePassword"  # Replace with the actual password

hex\_address = "0x1A2B3C4D5E6F"   # Replace with your hexadecimal meter address

latitude = "51.5074N"             # Replace with actual latitude

longitude = "0.1278W"             # Replace with actual longitude

# Build the transaction to call 'recordMeterData'

transaction = contract\_instance.functions.recordMeterData(

    password,

    hex\_address,

    latitude,

    longitude

).build\_transaction({

    "gasPrice": w3.eth.gas\_price,

    "chainId": 11155111,  # Example for Sepolia testnet (replace with the actual chain ID of your network)

    "from": wallet,

    "nonce": w3.eth.get\_transaction\_count(wallet)

})

# Sign the transaction with the private key of the wallet

signed\_txn = w3.eth.account.sign\_transaction(transaction, private\_key)

# Send the signed transaction to the network

tx\_hash = w3.eth.send\_raw\_transaction(signed\_txn.rawTransaction)

# Wait for the transaction receipt

tx\_receipt = w3.eth.wait\_for\_transaction\_receipt(tx\_hash)

# Print the transaction hash and receipt for verification

print(f"Transaction hash: {tx\_hash.hex()}")

print(f"Transaction receipt: {tx\_receipt}")