import socket

import time

import random

from web3 import Web3

provider = socket.socket()

print('Socket created')

provider.bind(('localhost', 9999))

provider.listen(3)

print('Waiting for connection')

# Connect to the Ethereum network and load the contract instance

web3 = Web3(Web3.HTTPProvider('http://localhost:7545'))

# Load the blockchain contract ABI and address

contract\_abi = [

    {

        "anonymous": False,

        "inputs": [

            {

                "indexed": True,

                "internalType": "uint256",

                "name": "requestId",

                "type": "uint256"

            },

            {

                "indexed": False,

                "internalType": "string",

                "name": "consumerName",

                "type": "string"

            },

            {

                "indexed": False,

                "internalType": "uint256",

                "name": "allocatedResources",

                "type": "uint256"

            }

        ],

        "name": "InteractionData",

        "type": "event"

    },

    {

        "inputs": [],

        "name": "interactionCount",

        "outputs": [

            {

                "internalType": "uint256",

                "name": "",

                "type": "uint256"

            }

        ],

        "stateMutability": "view",

        "type": "function"

    },

    {

        "inputs": [

            {

                "internalType": "uint256",

                "name": "",

                "type": "uint256"

            }

        ],

        "name": "interactions",

        "outputs": [

            {

                "internalType": "uint256",

                "name": "requestId",

                "type": "uint256"

            },

            {

                "internalType": "string",

                "name": "consumerName",

                "type": "string"

            },

            {

                "internalType": "uint256",

                "name": "allocatedResources",

                "type": "uint256"

            }

        ],

        "stateMutability": "view",

        "type": "function"

    },

    {

        "inputs": [

            {

                "internalType": "uint256",

                "name": "\_requestId",

                "type": "uint256"

            },

            {

                "internalType": "string",

                "name": "\_consumerName",

                "type": "string"

            },

            {

                "internalType": "uint256",

                "name": "\_allocatedResources",

                "type": "uint256"

            }

        ],

        "name": "storeInteraction",

        "outputs": [],

        "stateMutability": "nonpayable",

        "type": "function"

    }

]

contract\_address = '0xeeaf0F36D2a8ccD941138779544be65B6e49A17c'  # Address of your deployed contract

# Load the contract instance

contract = web3.eth.contract(address=contract\_address, abi=contract\_abi)

while True:

    consumer, addr = provider.accept()

    name = consumer.recv(1024).decode()

    print("Connected with", addr, name)

    consumer.send(bytes('Welcome', 'utf-8'))

    try:

        for \_ in range(10):

            try:

                # Request resources from the consumer

                consumer.send(bytes('request\_resources', 'utf-8'))

                # Receive the number of resources requested from the consumer

                resource\_request = consumer.recv(1024).decode()

                if resource\_request == 'request\_resources':

                    print("Waiting for resource request...")

                    continue  # Skip resource allocation if the consumer sent the request string

                try:

                    resource\_request = int(resource\_request)

                    print("Resource request:", resource\_request)

                except ValueError:

                    print("Invalid resource request received:", resource\_request)

                    continue  # Skip resource allocation if the consumer sent an invalid request

                # Randomly decide whether to allocate all requested resources or less

                if random.random() < 0.5:

                    allocated\_resources = min(resource\_request, 10)  # Allocate all or maximum of 10

                else:

                    allocated\_resources = random.randint(1, resource\_request)  # Allocate randomly between 1 and requested resources

                # Send the allocated resources to the consumer

                consumer.send(bytes(str(allocated\_resources), 'utf-8'))

                # Store the interaction on the blockchain

                interaction\_data = {

                    'requestId': int(time.time()),

                    'consumerName': name,

                    'allocatedResources': allocated\_resources

                }

                # Print the interaction data

                print("Interaction Data:")

                print(interaction\_data)

                time.sleep(1)  # Add a delay for demonstration purposes

            except ConnectionAbortedError:

                print("Connection with the consumer aborted.")

                break  # Break out of the for loop if the connection is aborted

    except ConnectionAbortedError:

        print("Connection with the consumer aborted.")

    except ConnectionResetError:

        print("Connection with the consumer reset.")

# Specify the "from" address

    from\_address = '0x38e87a1E66c85A98E74004aeE2D6137a16c1BBe9' # Update this with your desired Ethereum address

    # Send a transaction to the contract to store the interaction

    transaction = contract.functions.storeInteraction(

        interaction\_data['requestId'],

        interaction\_data['consumerName'],

        interaction\_data['allocatedResources']

    ).transact({'from': from\_address})

    consumer.close()

import socket

import random

consumer = socket.socket()

consumer.connect(('localhost', 9999))

name = input("Enter your name: ")

consumer.send(bytes(name, 'utf-8'))

# Receive the welcome message from the provider

welcome\_message = consumer.recv(1024).decode()

print(welcome\_message)

for \_ in range(10):

    # Generate a random number for resource request

    resource\_request = str(random.randint(1, 10))

    print("Requesting", resource\_request, "resources")

    # Send the resource request to the provider

    consumer.send(bytes(resource\_request, 'utf-8'))

    # Receive the allocated resources from the provider

    response = consumer.recv(1024).decode()

    if response == 'request\_resources':

        print("Waiting for resource allocation...")

        continue

    try:

        allocated\_resources = int(response)

        print("Allocated resources:", allocated\_resources)

    except ValueError:

        print("Invalid response received:", response)

consumer.close()