Creating a Smart Contract Factory to parse multiple Smart Contract Addresses & listening to real-time events from the private blockchain

Problem Faced:

In order to store multiple client’s data into the public blockchain, we would need to parse the smart contract address of each client into the node.js script. However, in order to update the smart contract of each client, we would need the smart contract address for each client. How do we parse the smart contract address of each client?

* Create the Policy Smart Contract ( contract PolicySmartContract ) in a file called clientfactory.sol

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.4.25 <0.9.0;

contract PolicySmartContract {

    //Emitted when update function is called

   //Smart contract events are a way for your contract to communicate that something happened on the blockchain to your app front-end, which can be 'listening' for certain events and take action when they happen.

   event \_updatedMessages(string newMessage, string policy, string duration, string date\_of\_policy, string phone, string name, string price);

   // Declares a state variable `message` of type `string`.

   // State variables are variables whose values are permanently stored in contract storage. The keyword `public` makes variables accessible from outside a contract and creates a function that other contracts or clients can call to access the value.

   string public message;

   string public policy;

   string public duration;

   string public date\_of\_policy;

   string public phone;

   string public name;

   string public price;

   // Similar to many class-based object-oriented languages, a constructor is a special function that is only executed upon contract creation.

   // Constructors are used to initialize the contract's data. Learn more:https://solidity.readthedocs.io/en/v0.5.10/contracts.html#constructors

   constructor(string memory newMessage, string memory \_policy, string memory \_duration, string memory \_date\_of\_policy, string memory \_phone, string memory \_name, string memory \_price) {

      message = newMessage;

      policy = \_policy;

      duration = \_duration;

      date\_of\_policy = \_date\_of\_policy;

      phone = \_phone;

      name = \_name;

      price = \_price;

   }

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

       return message;

   }

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

       return policy;

   }

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

       return duration;

   }

   function getDate\_of\_policy() view public returns (string memory) {

       return date\_of\_policy;

   }

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

       return phone;

   }

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

       return name;

   }

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

       return price;

   }

}

* In the same file clientfactory.sol, just below create a new contract called SmartContractFactory which will be in charge of the creation/deployment of multiple Policy Smart Contracts

contract SmartContractFactory {

    PolicySmartContract a;

    event Policy(PolicySmartContract a, string newMessage, string policy, string duration, string date\_of\_policy, string phone, string name, string price);

    function update(string memory newMessage, string memory \_policy, string memory \_duration, string memory \_date\_of\_policy, string memory \_phone, string memory \_name, string memory \_price) public returns(PolicySmartContract) {

      // Create the policy smart contract here

       a = new PolicySmartContract(newMessage, \_policy, \_duration, \_date\_of\_policy, \_phone, \_name, \_price);

       emit Policy(a, newMessage, \_policy, \_duration, \_date\_of\_policy, \_phone, \_name, \_price);

       return a;

   }

}

* Create a deployment script for the Smart Contract Factory called factorydeployment.js with the following contents

const { ethers } = require("hardhat");

async function main() {

    const Factory = await ethers.getContractFactory("SmartContractFactory");

    const factory = await Factory.deploy();

    console.log("Contract deployed to address:", factory.address);

 }

 main()

   .then(() => process.exit(0))

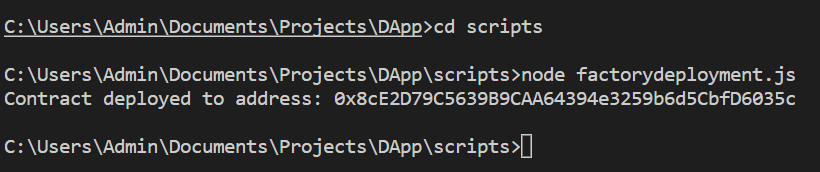
   .catch(error => {

     console.error(error);

     process.exit(1);

   });

* Run the Deployment Script and Copy the Smart Contract Address into a Notepad for later



* Create a new file called clientinforormation.js for the user to store and updated all their personal information into the already deployed Policy Smart Contract Factory. Include the following code; Inside the const contract, include the abi artifact link for the SmartContractFactory. Inside the const provider, include the websocket port connected to the private blockchain. Inside the signer, include one of the private keys from the genesis file in the private blockchain. And Inside the const Client, include the already deployed Policy Smart Contract Factory along with the abi and signer beside it.

const { ethers } = require("hardhat");

// For Hardhat

const contract = require("../artifacts/contracts/ClientFactory.sol/SmartContractFactory.json");

// console.log(JSON.stringify(contract.abi));

// Provider

const provider = new ethers.providers.WebSocketProvider("ws://localhost:9545");

// Signer

const signer = new ethers.Wallet("06b91f40bafe25bb97844ca576675d475ad4c197a4acab0f00a7dabf8326d480", provider);

// Contract

const Client = new ethers.Contract("0x3f21BC64076e7c9ed8695d053DCCBE6D8d5E6f43", contract.abi, signer);

async function main() {

  const prompt = require('prompt-sync')();

  const emailAddress = prompt('Input your Email Address:');

  console.log();

  const policy = prompt("Input your Policy ID: ");

  console.log();

  const duration = prompt("Input your duration: ");

  console.log();

  const date\_of\_policy = prompt("Input Date of Policy: ");

  console.log();

  const phone = prompt("Input Phone Number: ");

  console.log();

  const name = prompt("Input Full Name: ");

  console.log();

  const price = prompt("Input Price of Policy: ");

  console.log();

    const txc = await Client.update(`${emailAddress}`,`${policy}`,`${duration}`,`${date\_of\_policy}`,`${phone}`,`${name}`,`${price}`);

    const receipt = await txc.wait();

    console.log(receipt.events);

  }

  main();

* Last but not least, we need to create an event subscription to emit all of the clients information which has been successfully stored into the Policy Smart Contract Factory in real time on a separate js file. This file can be named as retrieveclientinfo.js. Include the following code;

const { ethers } = require("hardhat");

// For Hardhat

const contract = require("../artifacts/contracts/ClientFactory.sol/SmartContractFactory.json");

// console.log(JSON.stringify(contract.abi));

// Provider

const provider = new ethers.providers.WebSocketProvider("ws://localhost:9545");

// Signer

const signer = new ethers.Wallet("06b91f40bafe25bb97844ca576675d475ad4c197a4acab0f00a7dabf8326d480", provider);

// Contract

const Client = new ethers.Contract("0x3f21BC64076e7c9ed8695d053DCCBE6D8d5E6f43", contract.abi, signer);

async function main() {

    abi = ["event Policy(PolicySmartContract a, string newMessage, string policy, string duration, string date\_of\_policy, string phone, string name, string price);"];

    console.log("Waiting for event");

    Client.on('Policy', (a, newMessage, policy, duration, date\_of\_policy, phone, name, price) => {

      console.log("Smart Contract Address:", a)

      console.log("Email Address:", newMessage)

      console.log("Policy ID:", policy)

      console.log("Duration of Policy:", duration)

      console.log("Date of Policy:", date\_of\_policy)

      console.log("Phone number:", phone)

      console.log("Full Name:", name)

      console.log("Price:", price)

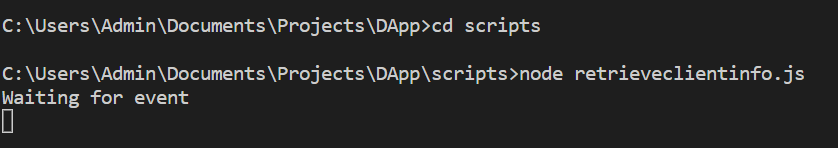
      console.log("................")

  })

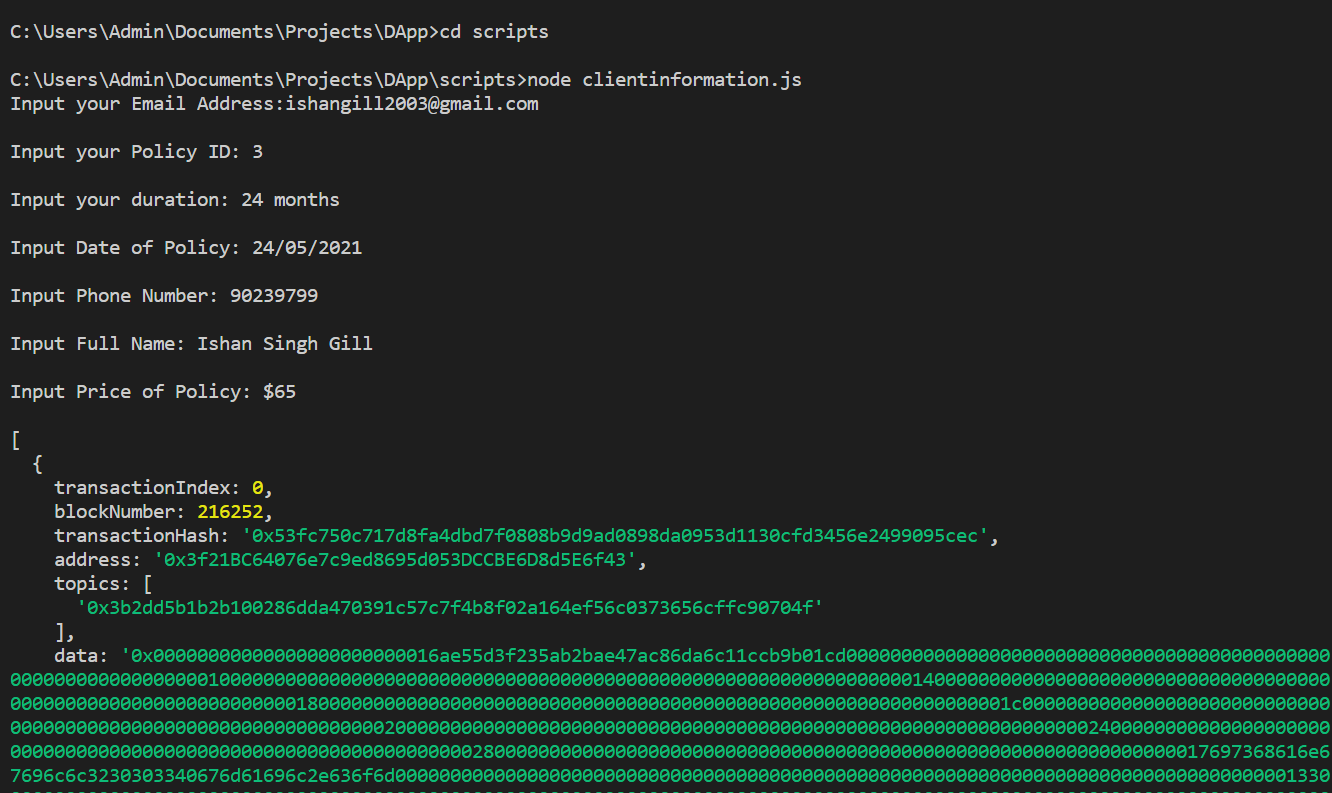
}

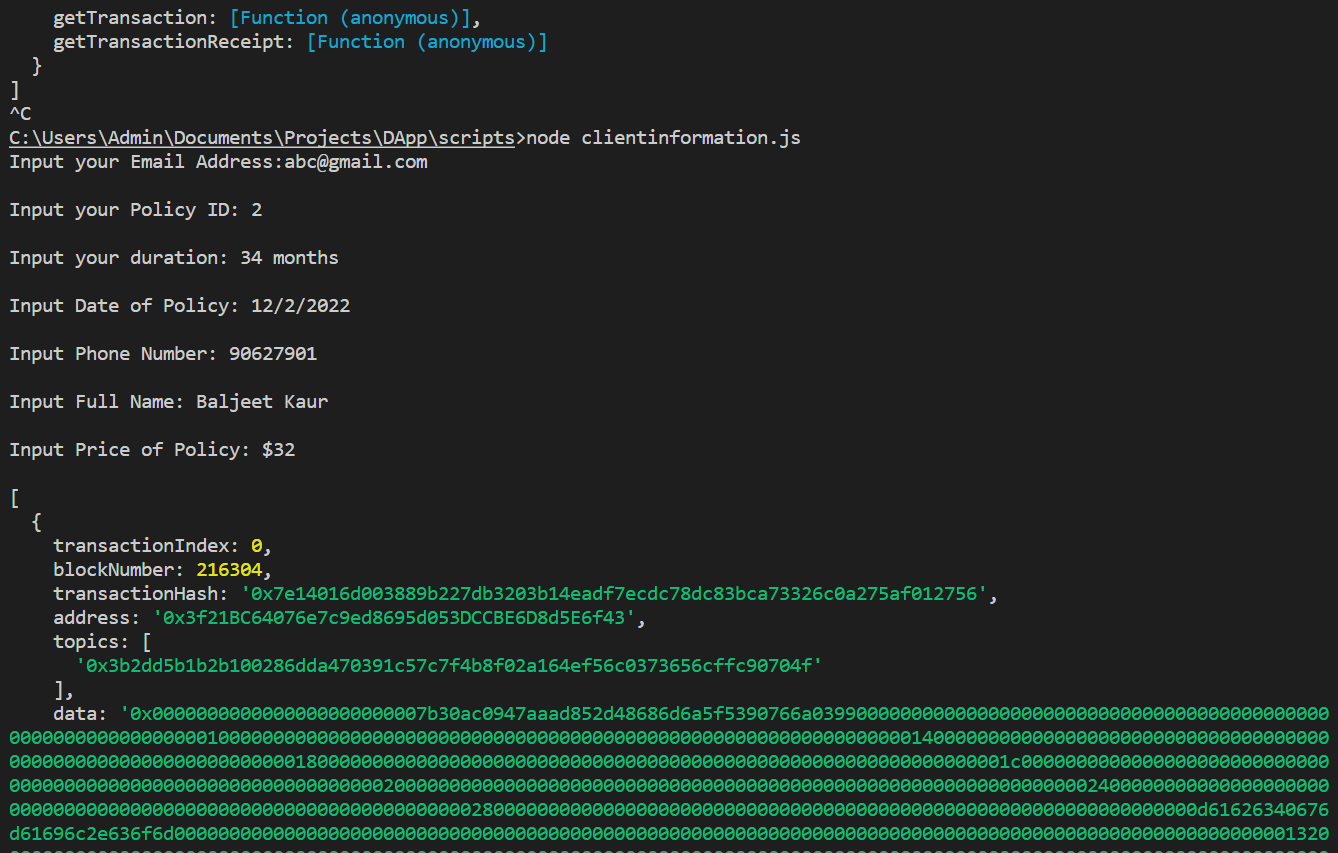
  main();

* Run both of the Node.js Files
  + Run the retrieveclientinfo.js file on command prompt



* + Run the clientinformation.js file twice on command prompt





* + Check the retrieveclientinfo.js file on command prompt

