目录

[1. Web3 Integration 2](#_Toc15418)

[1.1 2 ways to interact with blockchain 2](#_Toc31199)

[1.2 query data 4](#_Toc29880)

[1.3 add web3 to the frontend 4](#_Toc22338)

[1.4 add IPFS 6](#_Toc11027)

[2 Write a Store Factory (no Proxy) 6](#_Toc8001)

[1coding 7](#_Toc3589)

[Step 1 start ganache 7](#_Toc20800)

[2 Manual test 8](#_Toc17577)

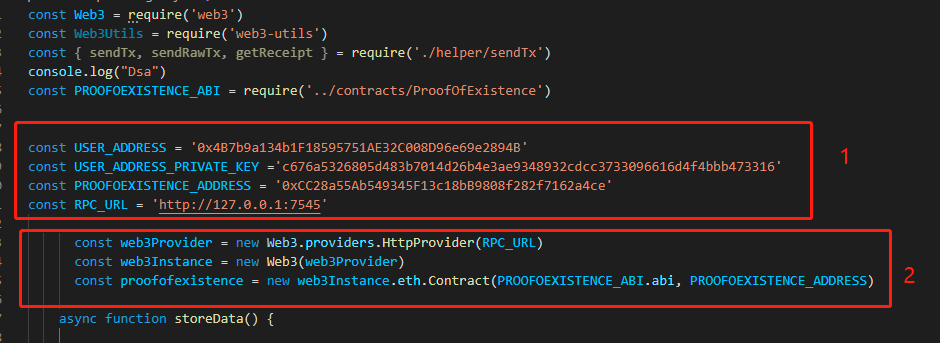
[3 make privateStamp factory upgradeable 8](#_Toc12303)

[3.1Verify old logic 8](#_Toc26156)

[3.2 Upgrade to v2 9](#_Toc16217)

# Web3 Integration

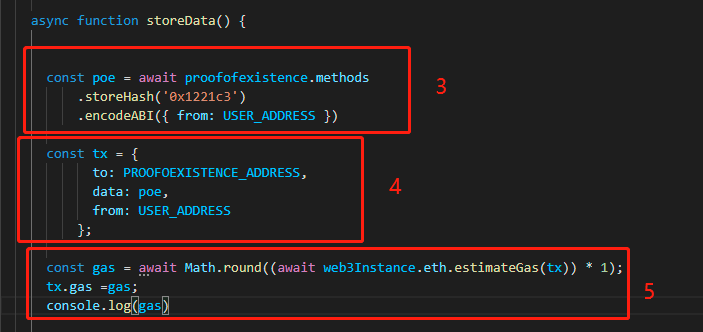
## 1.1 2 ways to interact with blockchain



Step1 : set addresses

Step2: create web3 connector

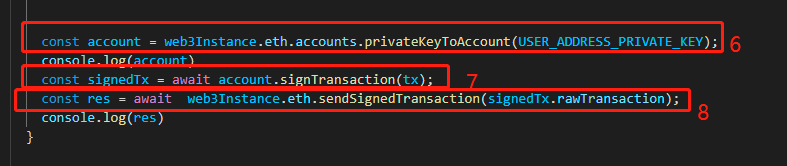
1 web3 send transaction



Step3:get method encode

Step4:assign encode and other params to tx

Step5:get estimate gas

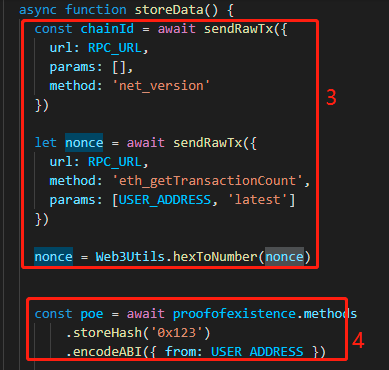


Step6:import web3 eth account from private key

Step7:sign the raw data with pk

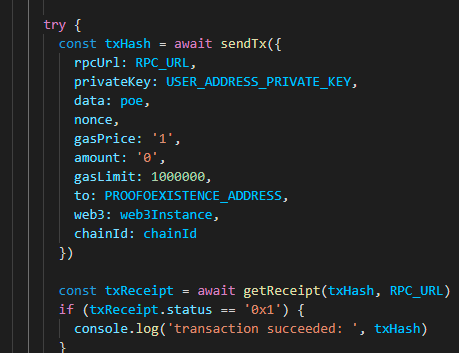
Step8: send signed transaction to blockchain

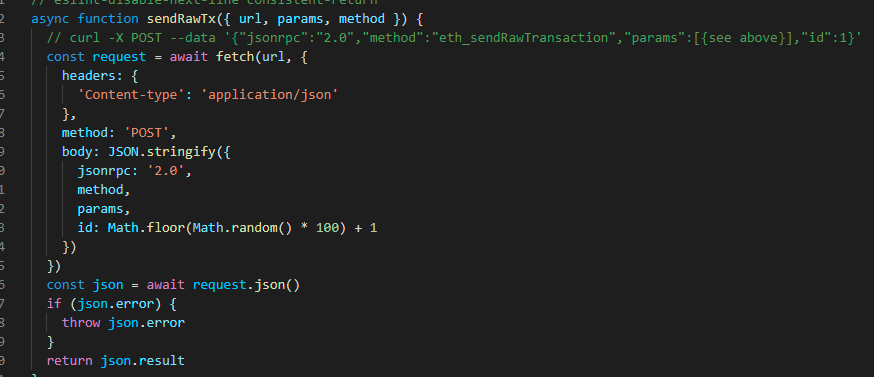
2 RPC post request



Step3: get chainid and nonce

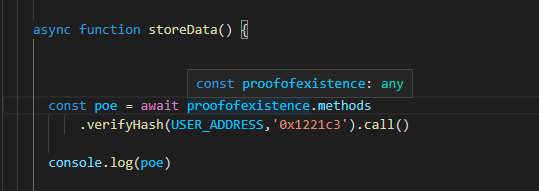
Step4:get methods encode





Use rpc post api to send signed raw data

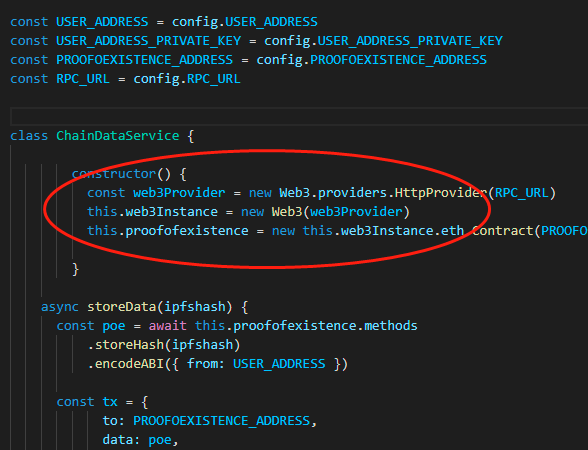
## 1.2 query data



Use .call()

## 1.3 add web3 to the frontend

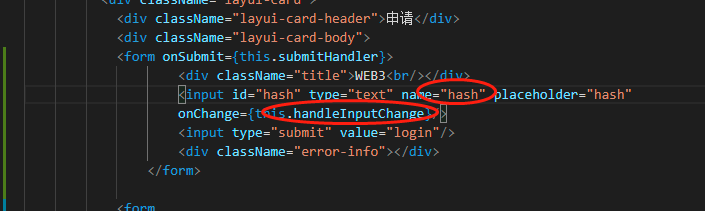
Step 1: convert the script to class



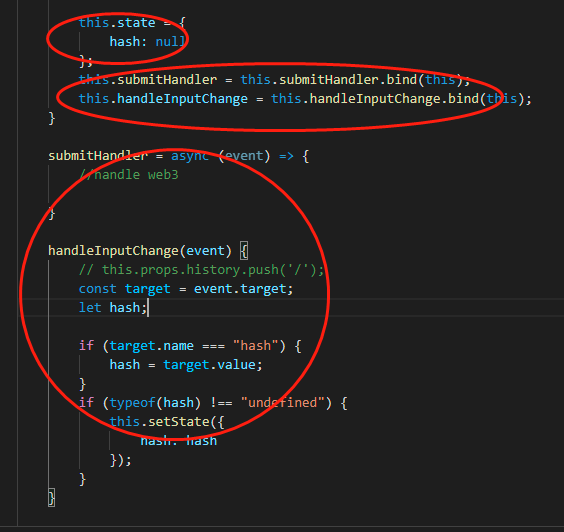
Make web3connector part to be the constructor

And put the rest of the code from the scripts to the storeData function

Step2: add input button to the front end



Step3: add event handlers

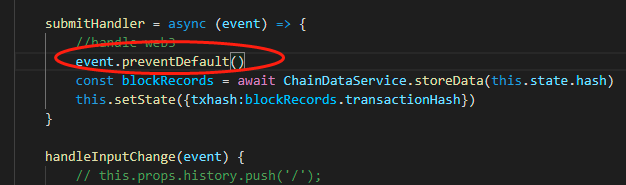


Set state

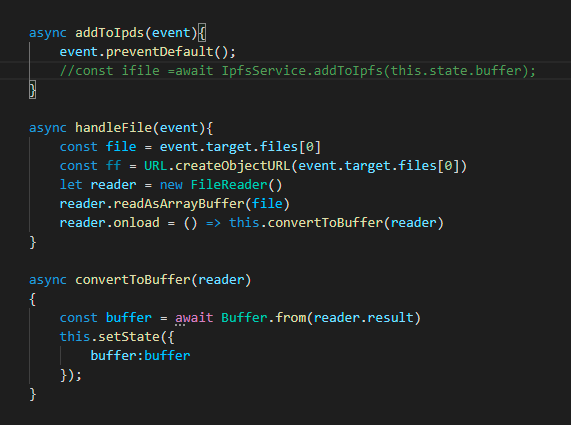
Set event handler

Bind this

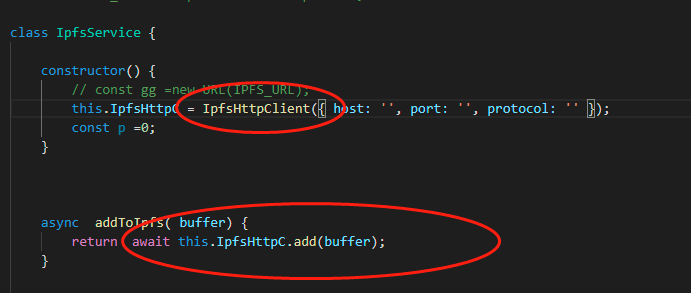
Step 4: add submit and web3 interaction



## 1.4 add IPFS



Store files in buffer state variable



Upload buffer to ipfs-client

# 2 Write a Store Factory (no Proxy)

Github.com/onebit256/proxy

## 1coding

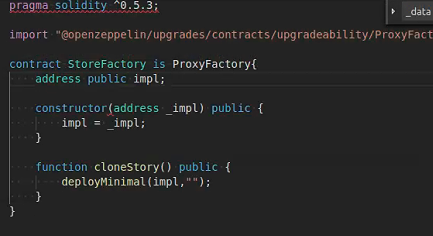
## Step 1 start ganache

Ganache-cli -m ‘’

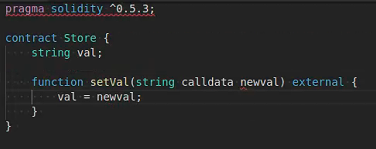
Truffle init

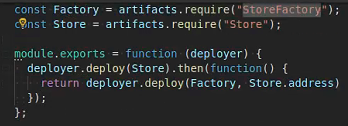
Npm init

Npm i @openzeppelin/upgrades@2.8.0



As there is no init function





If there are init data



## 2 Manual test

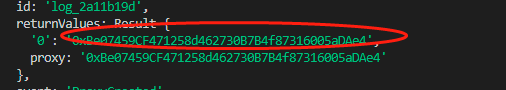
Const sf = await StoreFactory.deployed()

Sf.impl()

Networks

Sf.clonestore()

Sf.getPastEvents()



Const St1 = await Store.at(‘0xBe07459CF471258d462730B7B4f87316005aDAe4’)

St1.setVal(“ss”)

St1.val()

Sf.clonestore()

Sf.getPastEvents()

Const St2 = await Store.at(‘0xBe07459CF471258d462730B7B4f87316005aDAe4’)

St2.setVal(“ss”)

St2.val()

Sf.clonestore()

# 3 make privateStamp factory upgradeable

## 3.1Verify old logic

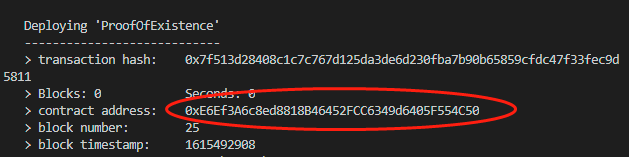
const proxyFactory =await ProxyFactory.at(ProxyFactory.address)

const proxyAdmin =await ProxyAdmin.at(ProxyAdmin.address)

const salt = "20";

const deploymentProxyAddress = await proxyFactory.getDeploymentAddress(salt, accounts[0]);

const implementationAddress = await proxyAdmin.getProxyImplementation(deploymentProxyAddress);





const poe1 =await ProofOfExistence.at(deploymentProxyAddress)

poe1.storeHash("123")

poe1.verifyHash(accounts[0],"123")



## 3.2 Upgrade to v2

await proxyAdmin.upgrade(deploymentProxyAddress, ProofOfExistenceV2.address)



const poe2 = await ProofOfExistenceV2.at(deploymentProxyAddress)

