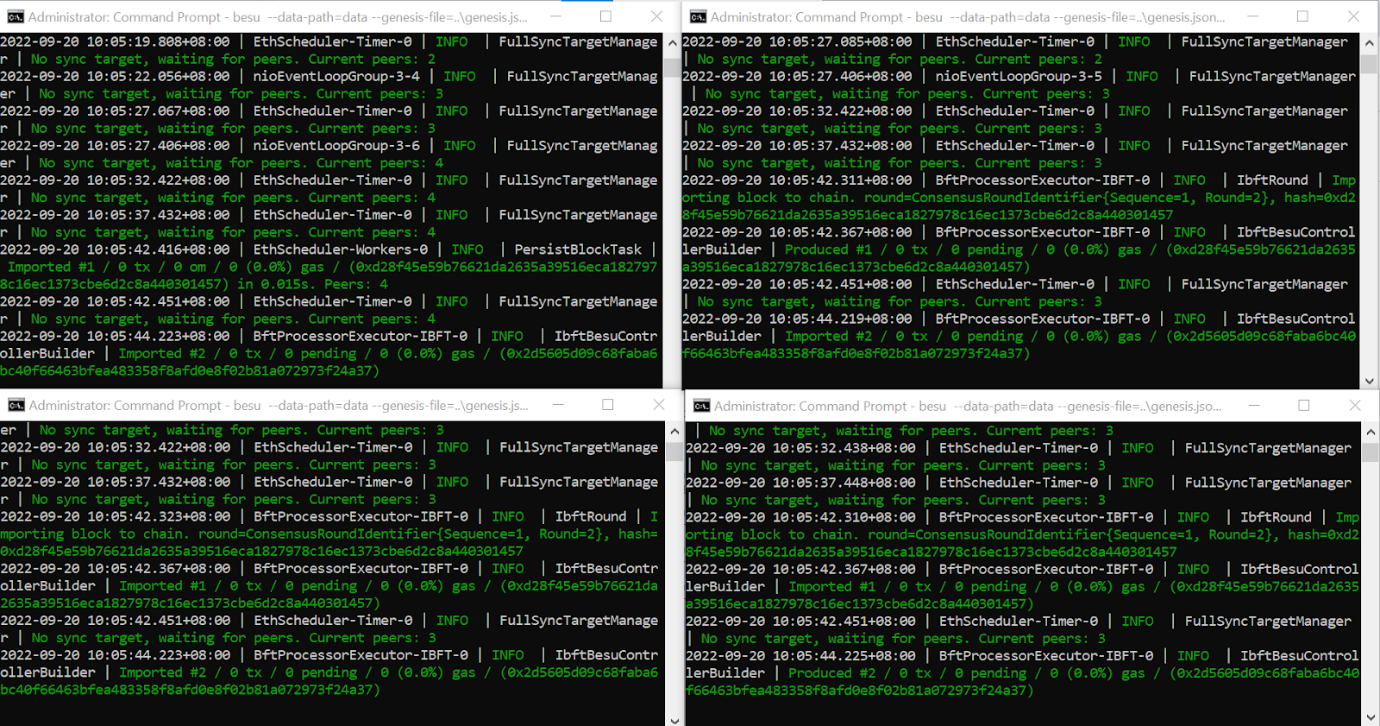
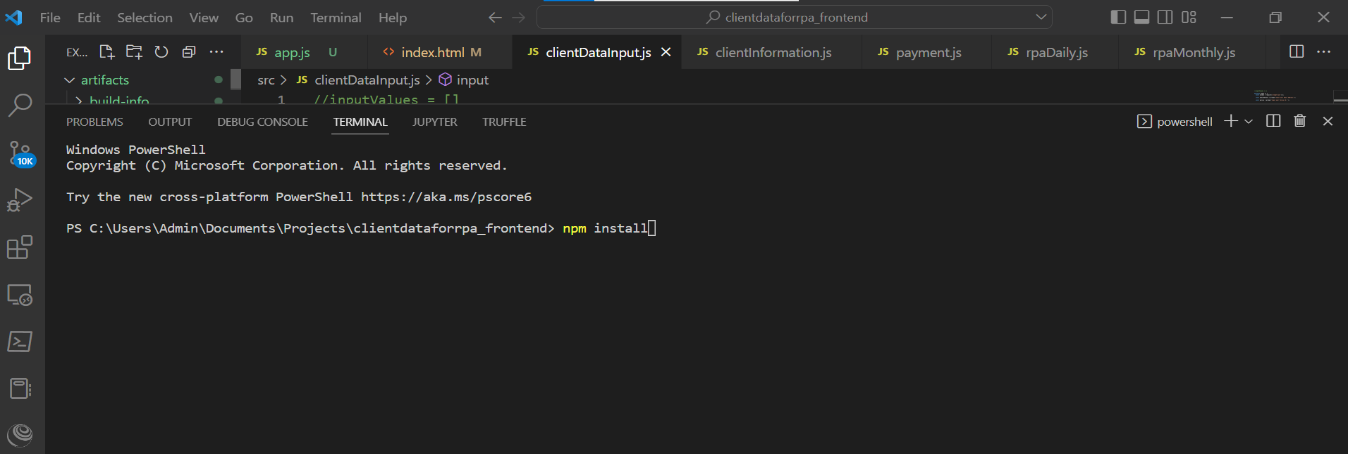
Presentation Demo Setup

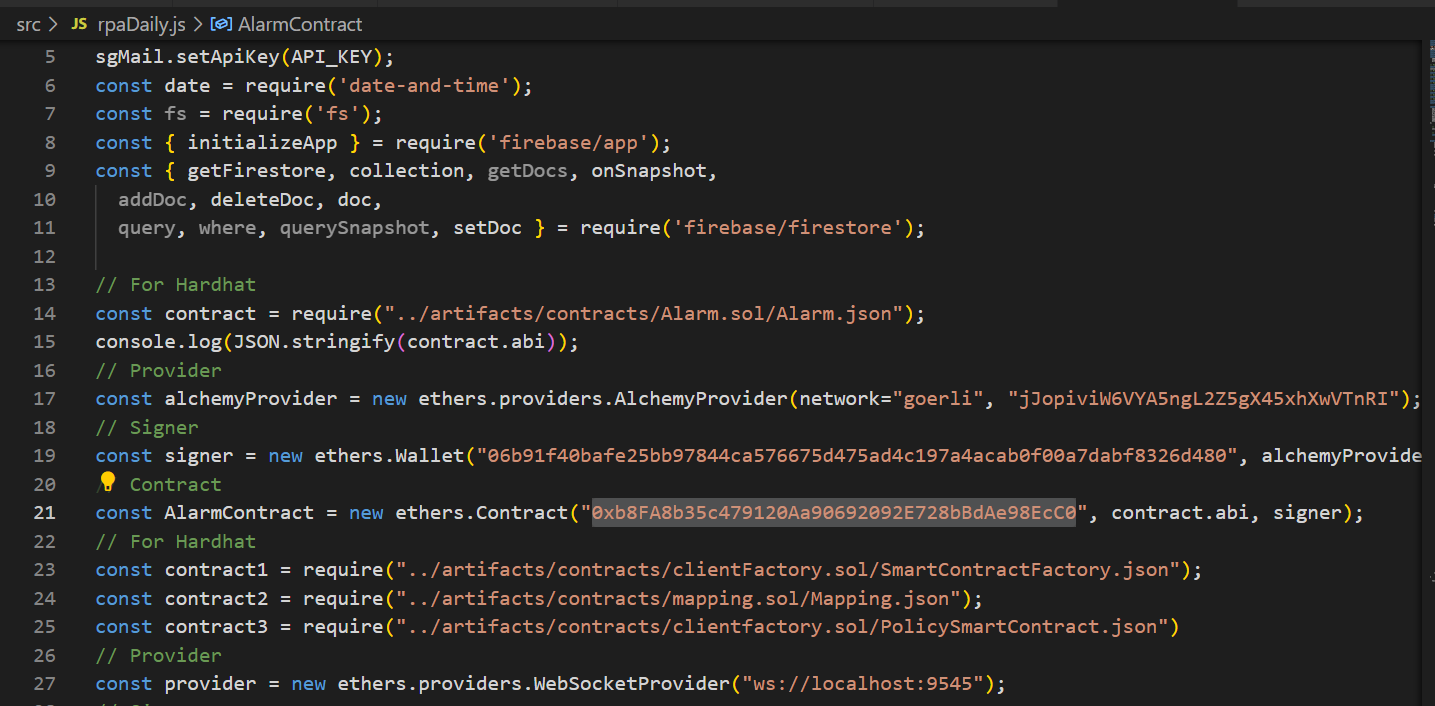
* Run the Private Blockchain with the following commands
  + Node 1 - besu --data-path=data --genesis-file=..\genesis.json --rpc-http-host=0.0.0.0 --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT,EEA,PRIV --host-allowlist="\*" --rpc-http-cors-origins="all" --privacy-enabled --privacy-url=http://127.0.0.1:9102 --privacy-public-key-file=Tessera\nodeKey.pub --min-gas-price=0 --rpc-ws-enabled=true --rpc-ws-host=0.0.0.0 --rpc-ws-port=9545
  + Node 2 - besu --data-path=data --genesis-file=..\genesis.json --bootnodes=enode://a32b7b959c0acc4fda78814f1261ab964b535a7e63f25f9fd149b391a016297c7f8b200b9e8a90790904ac0e68ff9c31c7fae63dca866a2e4080a314c7bebe86@127.0.0.1:30303 --p2p-port=30304 --rpc-http-host=0.0.0.0 --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT,EEA,PRIV --host-allowlist="\*" --rpc-http-cors-origins="all" --rpc-http-port=8546 --privacy-enabled --privacy-url=http://127.0.0.1:9202 --privacy-public-key-file=Tessera\nodeKey.pub --min-gas-price=0 --rpc-ws-enabled=true --rpc-ws-host=0.0.0.0 --rpc-ws-port=9546
  + Node 3 - besu --data-path=data --genesis-file=..\genesis.json --bootnodes=enode://a32b7b959c0acc4fda78814f1261ab964b535a7e63f25f9fd149b391a016297c7f8b200b9e8a90790904ac0e68ff9c31c7fae63dca866a2e4080a314c7bebe86@127.0.0.1:30303 --p2p-port=30305 --rpc-http-host=0.0.0.0 --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT,EEA,PRIV --host-allowlist="\*" --rpc-http-cors-origins="all" --rpc-http-port=8547 --privacy-enabled --privacy-url=http://127.0.0.1:9302 --privacy-public-key-file=Tessera\nodeKey.pub --min-gas-price=0 --rpc-ws-enabled=true --rpc-ws-host=0.0.0.0 --rpc-ws-port=9547
  + Node 4 - besu --data-path=data --genesis-file=..\genesis.json --bootnodes=enode://a32b7b959c0acc4fda78814f1261ab964b535a7e63f25f9fd149b391a016297c7f8b200b9e8a90790904ac0e68ff9c31c7fae63dca866a2e4080a314c7bebe86@127.0.0.1:30303 --p2p-port=30306 --rpc-http-host=0.0.0.0 --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT,EEA,PRIV --host-allowlist="\*" --rpc-http-cors-origins="all" --rpc-http-port=8548 --privacy-enabled --privacy-url=http://127.0.0.1:9402 --privacy-public-key-file=Tessera\nodeKey.pub --min-gas-price=0 --rpc-ws-enabled=true --rpc-ws-host=0.0.0.0 --rpc-ws-port=9548

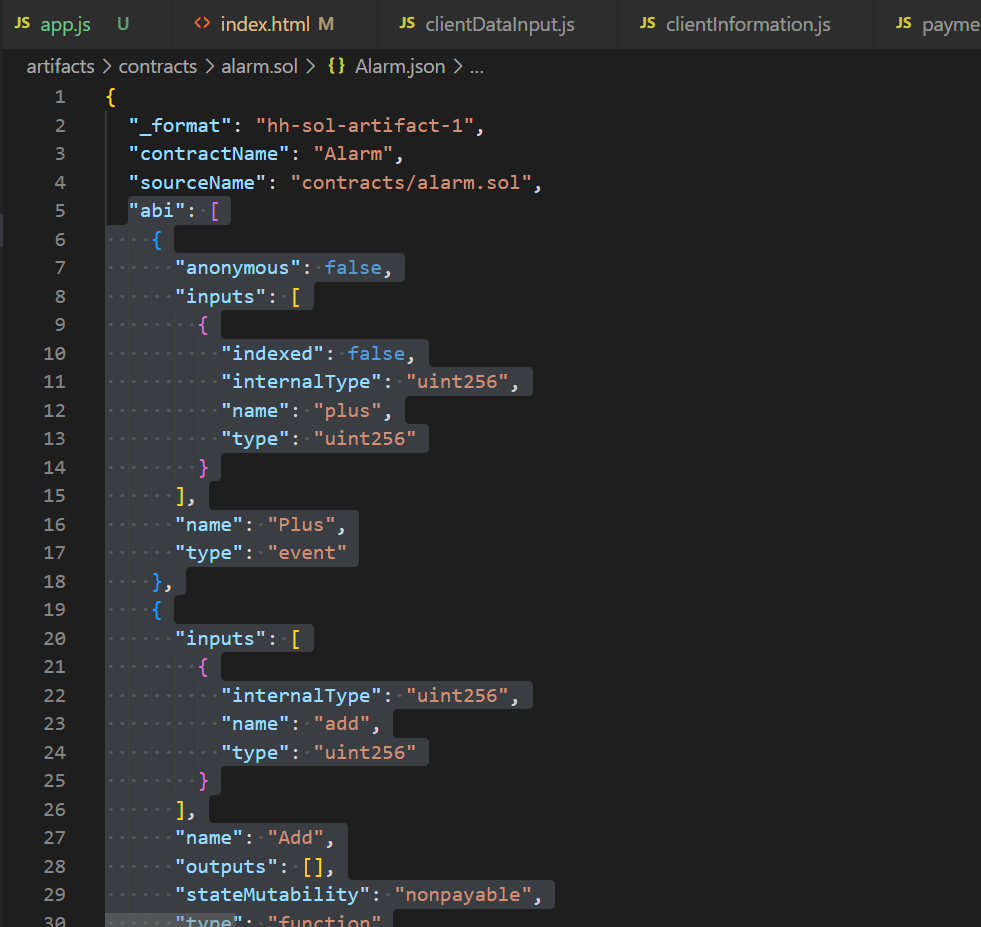


* Setup the Gelato Network
  + Open up the clientdataforrpa\_frontend folder on VsCode and run “npm install”

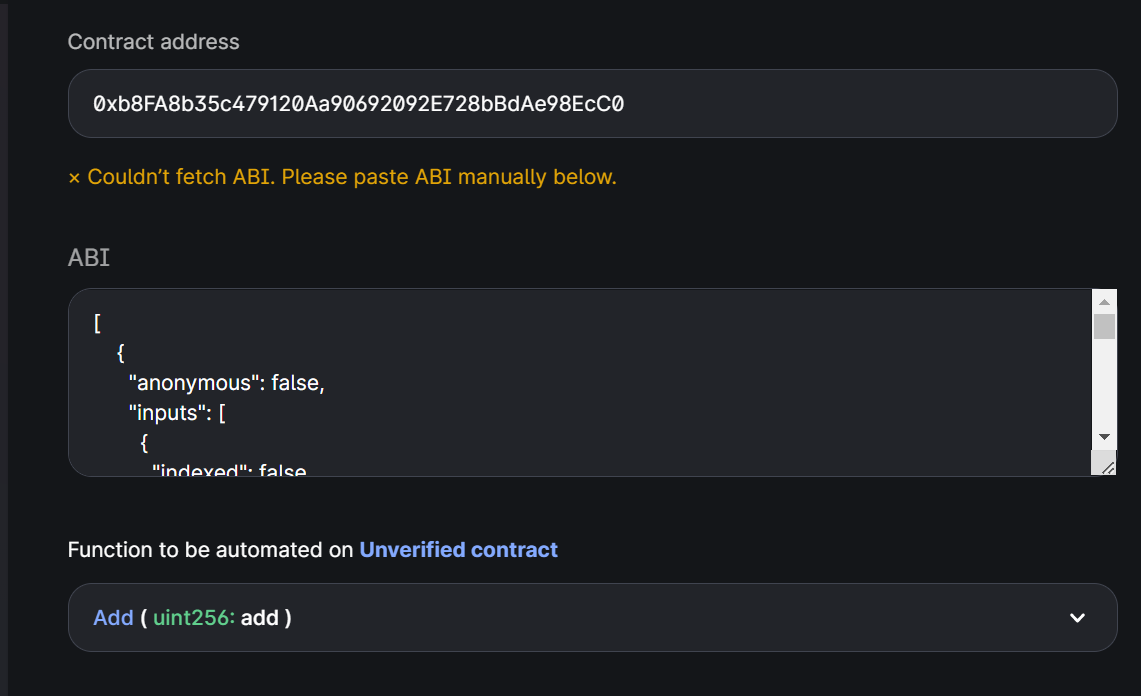


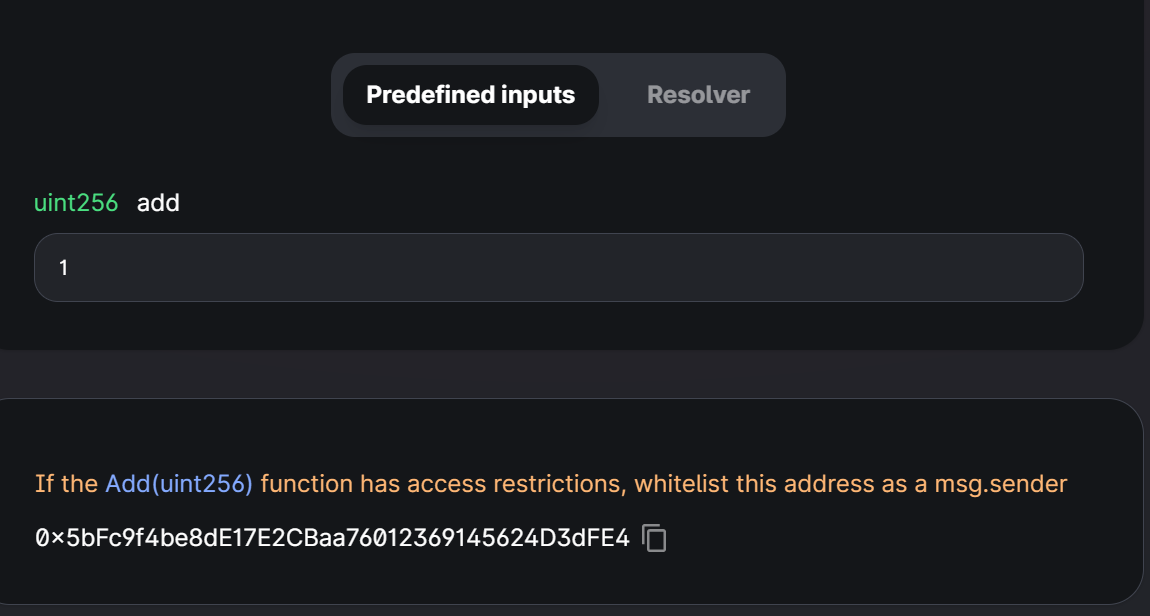
* + Head to rpaDaily.js and copy the contract address and abi of alarm.sol

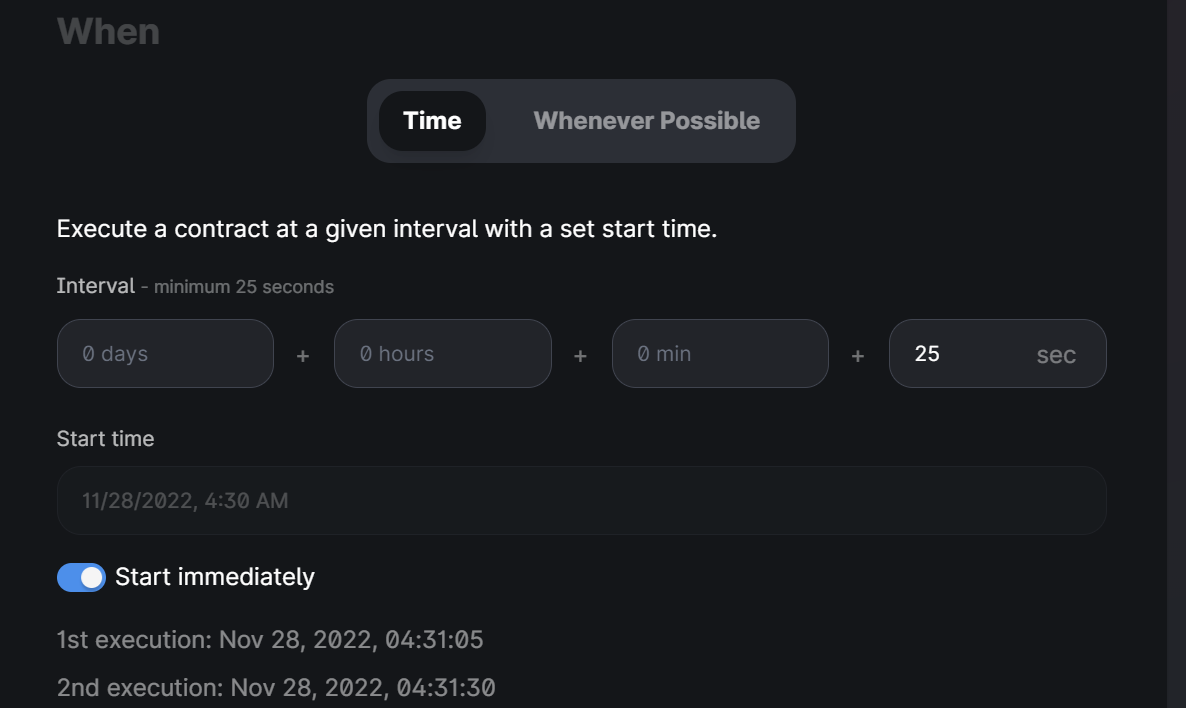


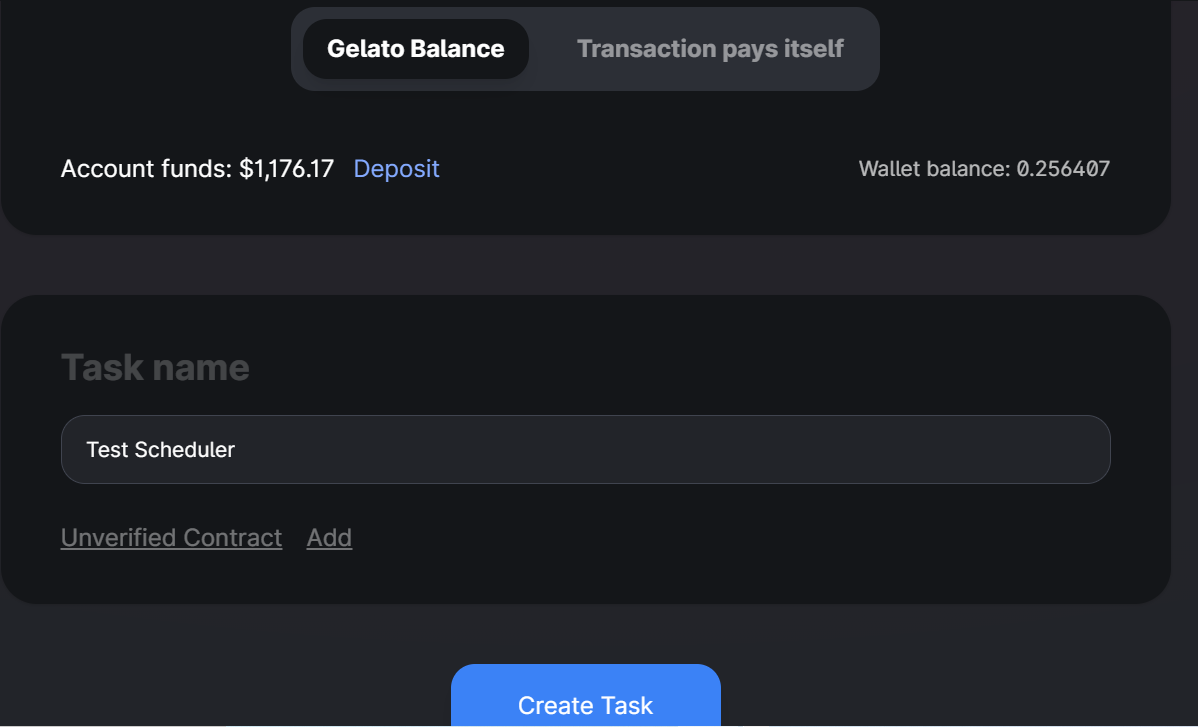


* + Head to the Gelato Automation Website (<https://app.gelato.network/>) and create a new Task with the contract address and the following details

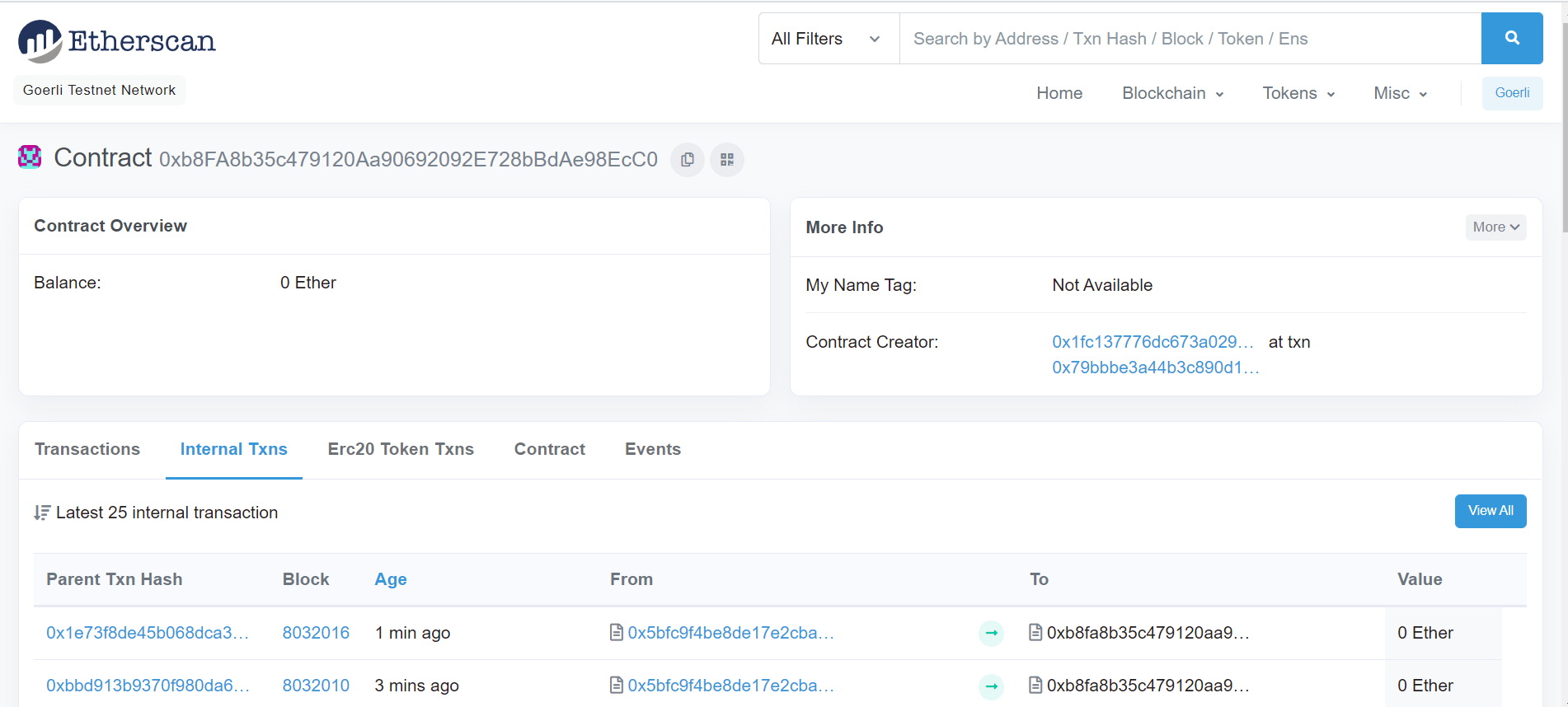




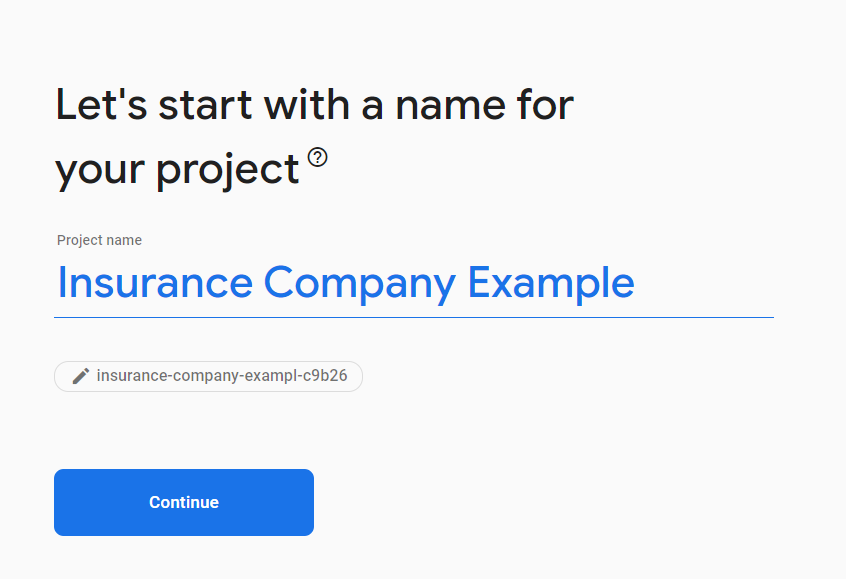


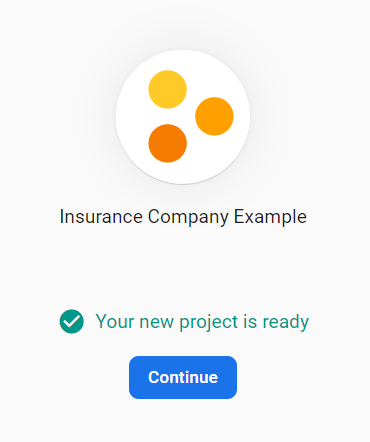


* + Once its executed click on the contract Address and check for the internal transactions under Goreli Etherscan

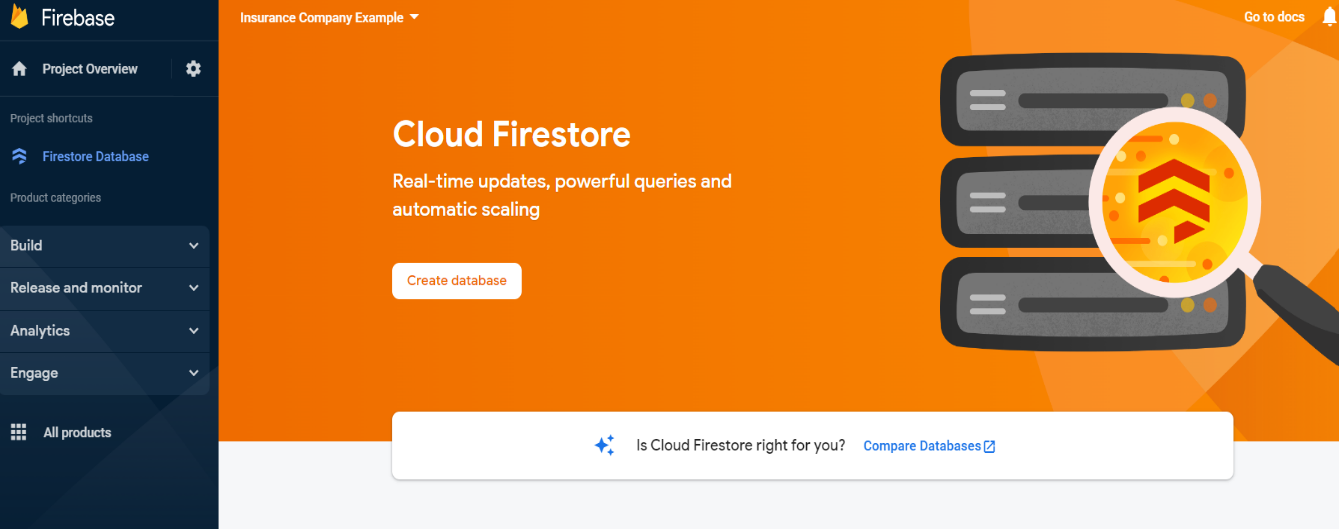


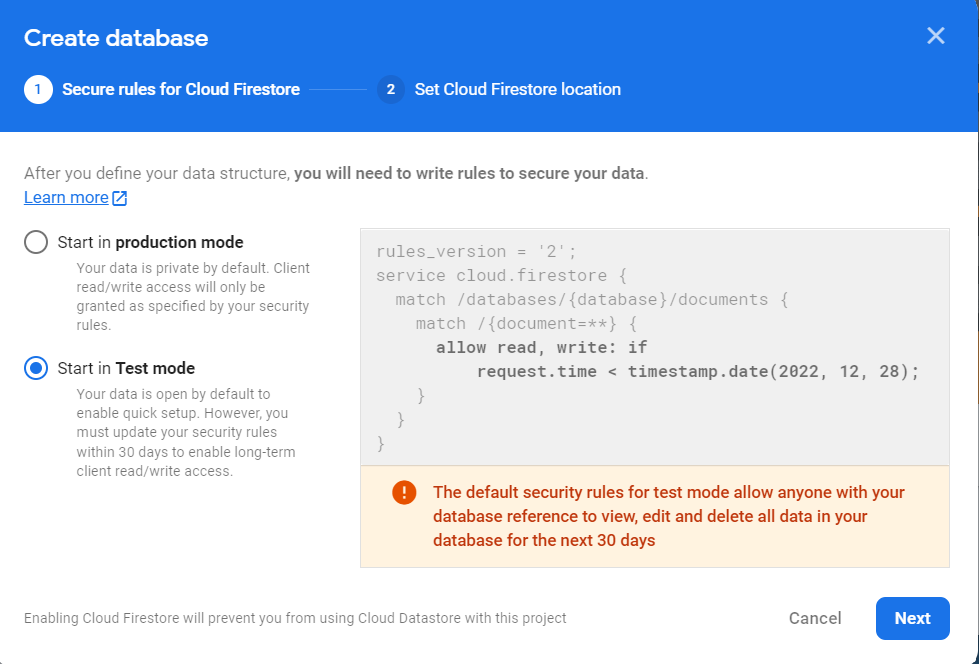
* Setup the Firebase Database
  + Click on Go to Console and Create a New Project with the name Insurance Company (Rest of the Options are left on default)

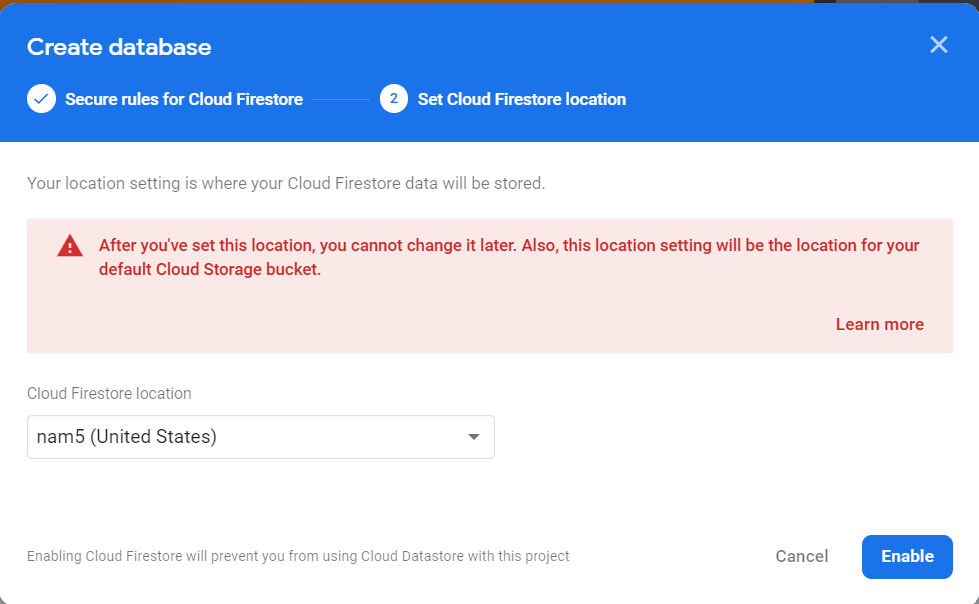


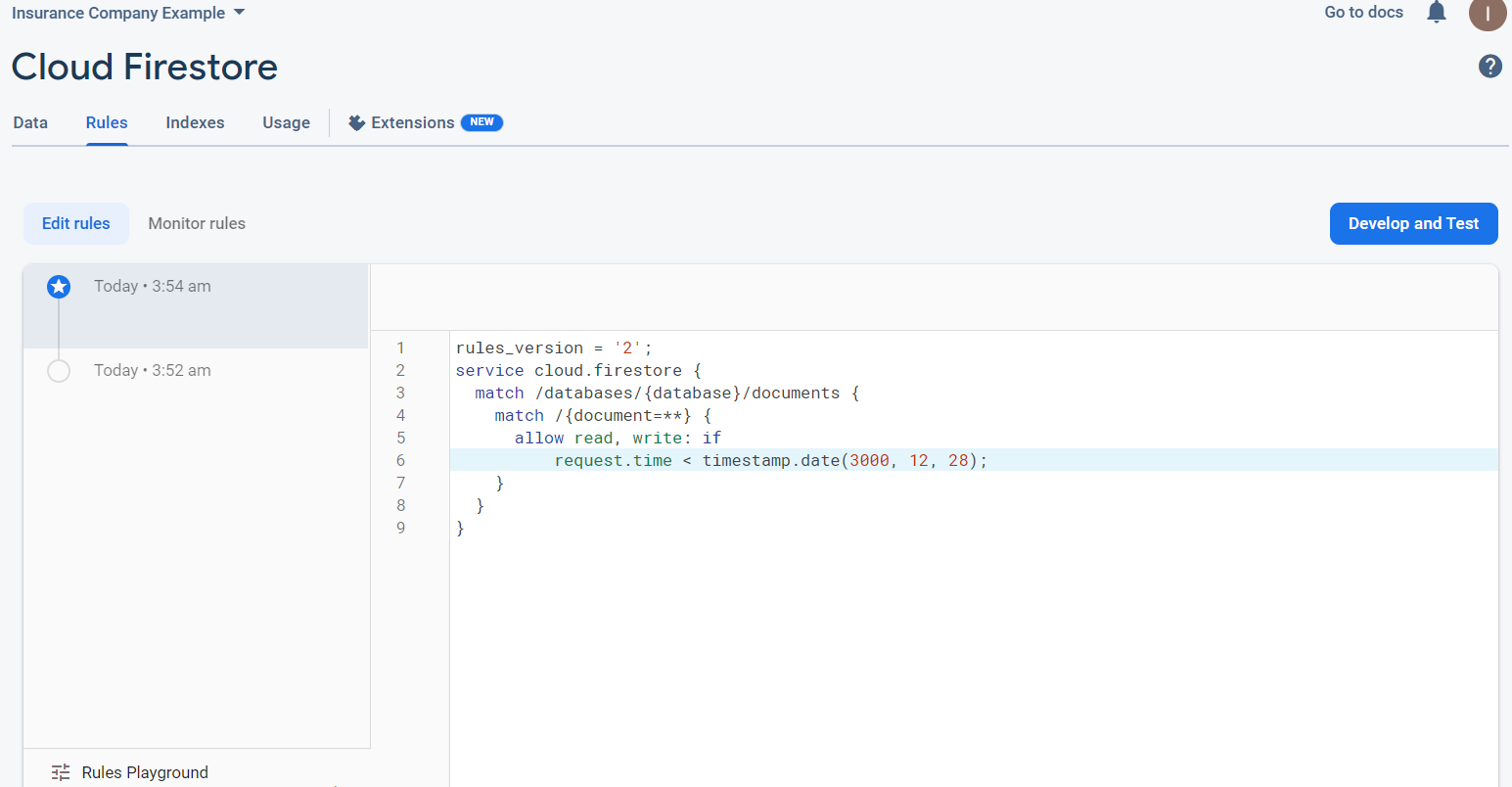


* + Click on Firestore Database and create a new Database with the following options and rules

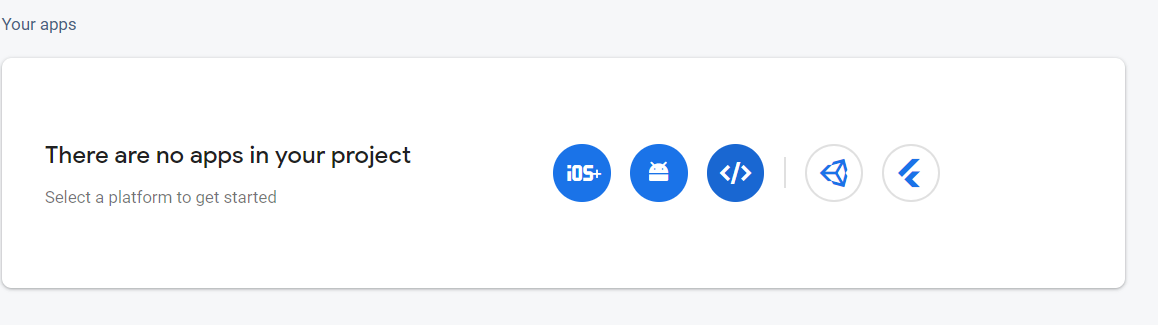


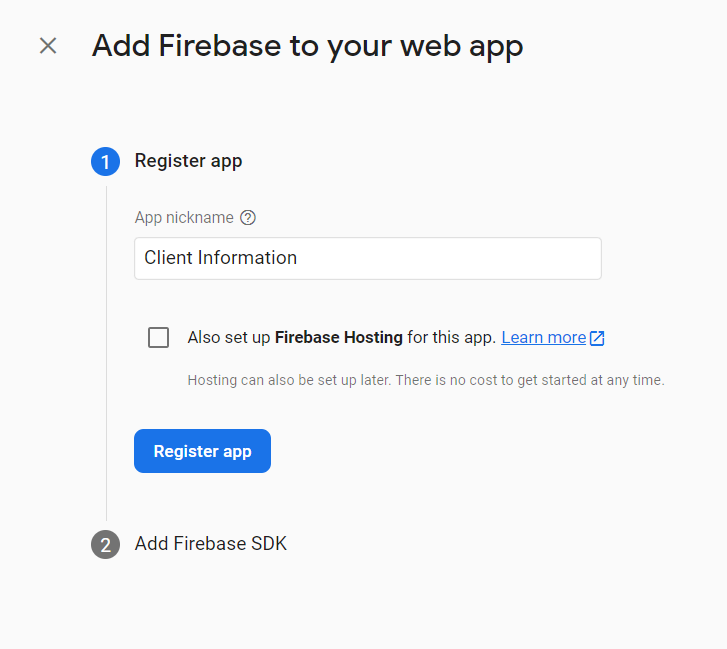


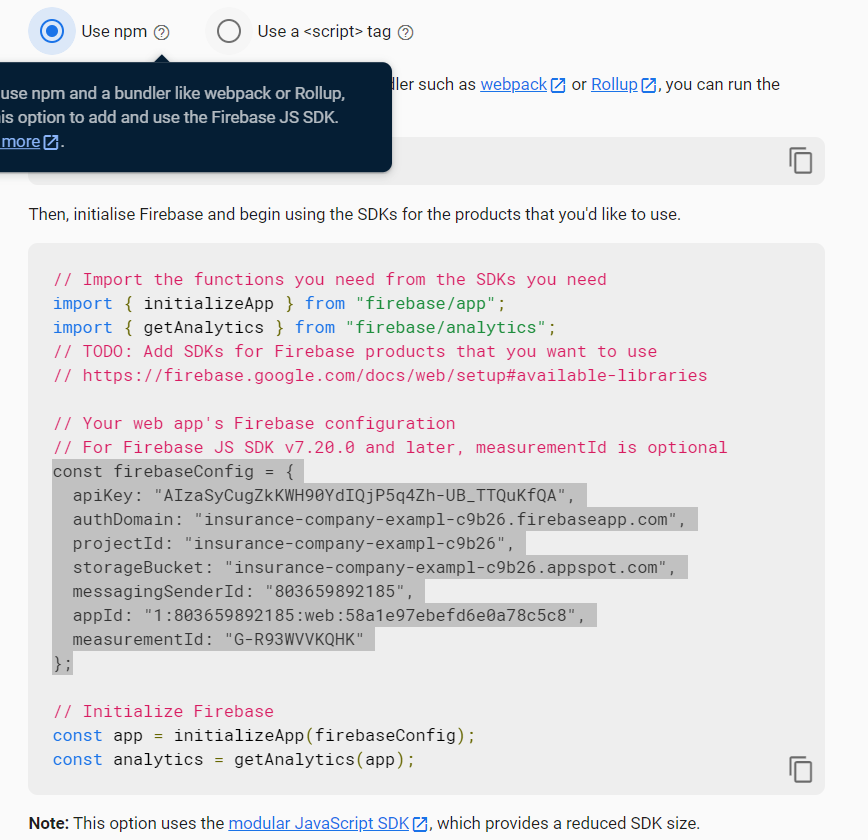


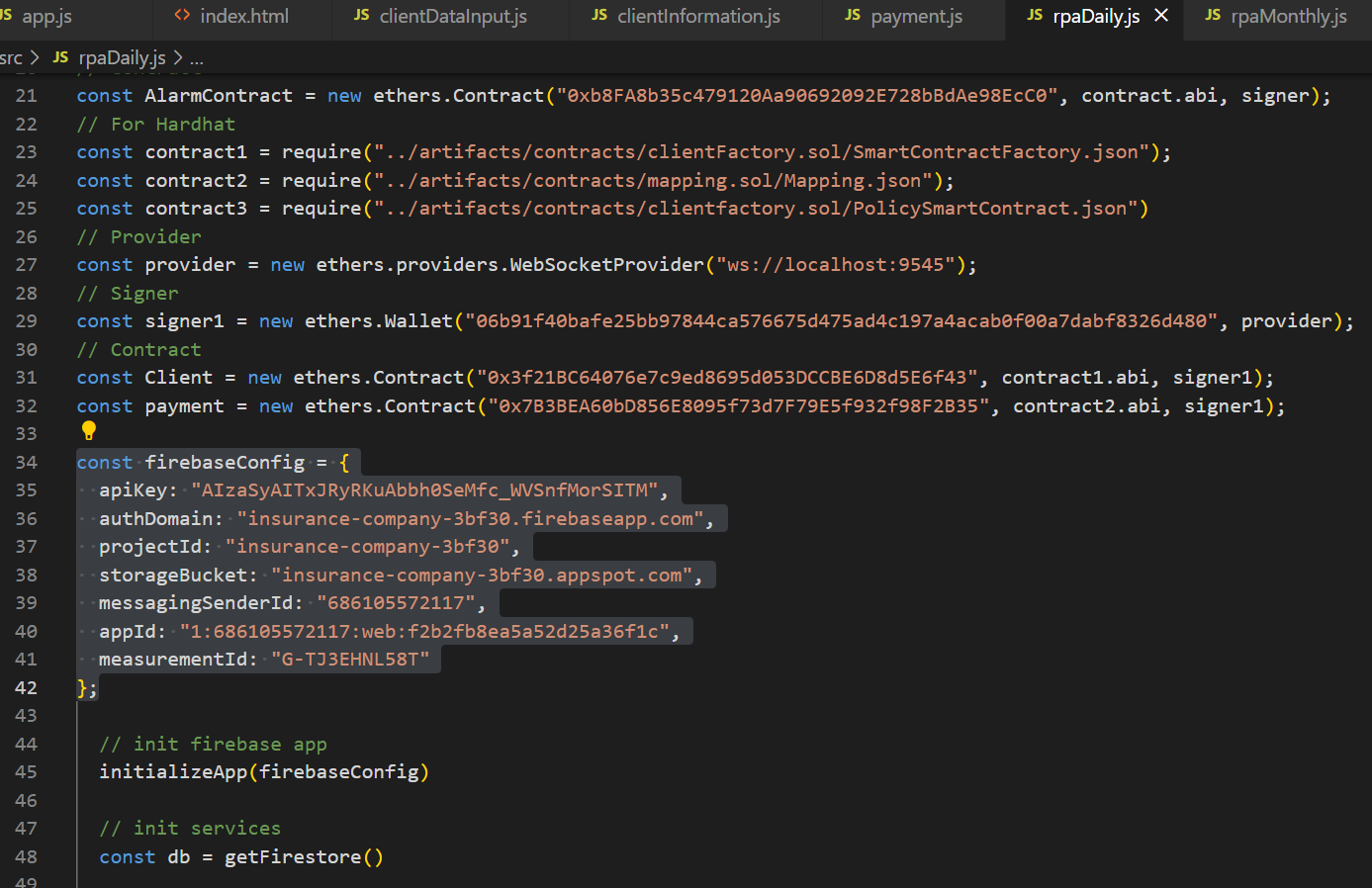


* + Click on Project Settings to Copy the Firebase Config and paste it into rpaDaily.js in the clientforrpa\_frontend folder in VsCode

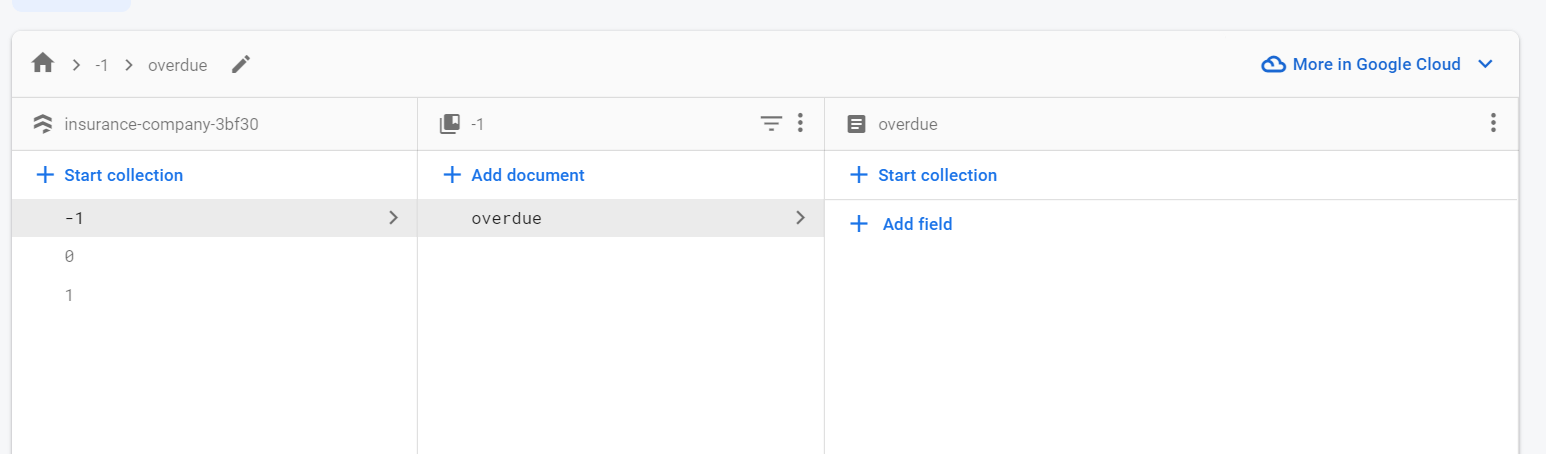








* + Create the Database Structure in the form:
    - -1 (Collection) => overdue (document id)
    - 0 (Collection) => unpaid (document id)
    - 1 (Collection) => paid (document id)



* Code the Following Backend Files in the following order
  + Create a mapping smart contract call mapping.sol (under the contracts folder) and paste the following code

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.4.25 <0.9.0;

contract Mapping {

    // Mapping from address to uint

    mapping(string => address) public myMap;

    event Map(address contractAddress);

    function getId(string memory \_id) public returns (address) {

        // Mapping always returns a value.

        // If the value was never set, it will return the default value.

        emit Map(myMap[\_id]);

        return myMap[\_id];

    }

    function getIdview(string memory \_id) public view returns (address) {

       return myMap[\_id];

    }

    function setId(string memory \_id, address \_addr) public {

        // Update the value at this address

        myMap[\_id] = \_addr;

    }

    function removeId(string memory \_id) public {

        // Reset the value to the default value.

        delete myMap[\_id];

    }

}

* + Write the deploy scripts for all 3 smart contracts and deploy 2 of them on the private blockchain and 1 on the public blockchain
    - Write the deploy script (deploy.js) for the alarm.sol smart contract and deploy it to the goreli testnet (public blockchain)

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

async function main() {

    const Alarm = await ethers.getContractFactory("Alarm");

    const alarm = await Alarm.deploy();

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

 }

 main()

   .then(() => process.exit(0))

   .catch(error => {

     console.error(error);

     process.exit(1);

   });

* + - Write the deploy script (deployBesu.js) for the mapping.sol smart contract and deploy it to Hyperledger Besu (private blockchain)

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

async function main() {

    const Mapping = await ethers.getContractFactory("Mapping");

    const mapping = await Mapping.deploy();

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

 }

 main()

   .then(() => process.exit(0))

   .catch(error => {

     console.error(error);

     process.exit(1);

   });

* + - Write the deploy script (deployFactory.js) for the clientFactory.sol smart contract and deploy it to Hyperledger Besu (private blockchain)

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))

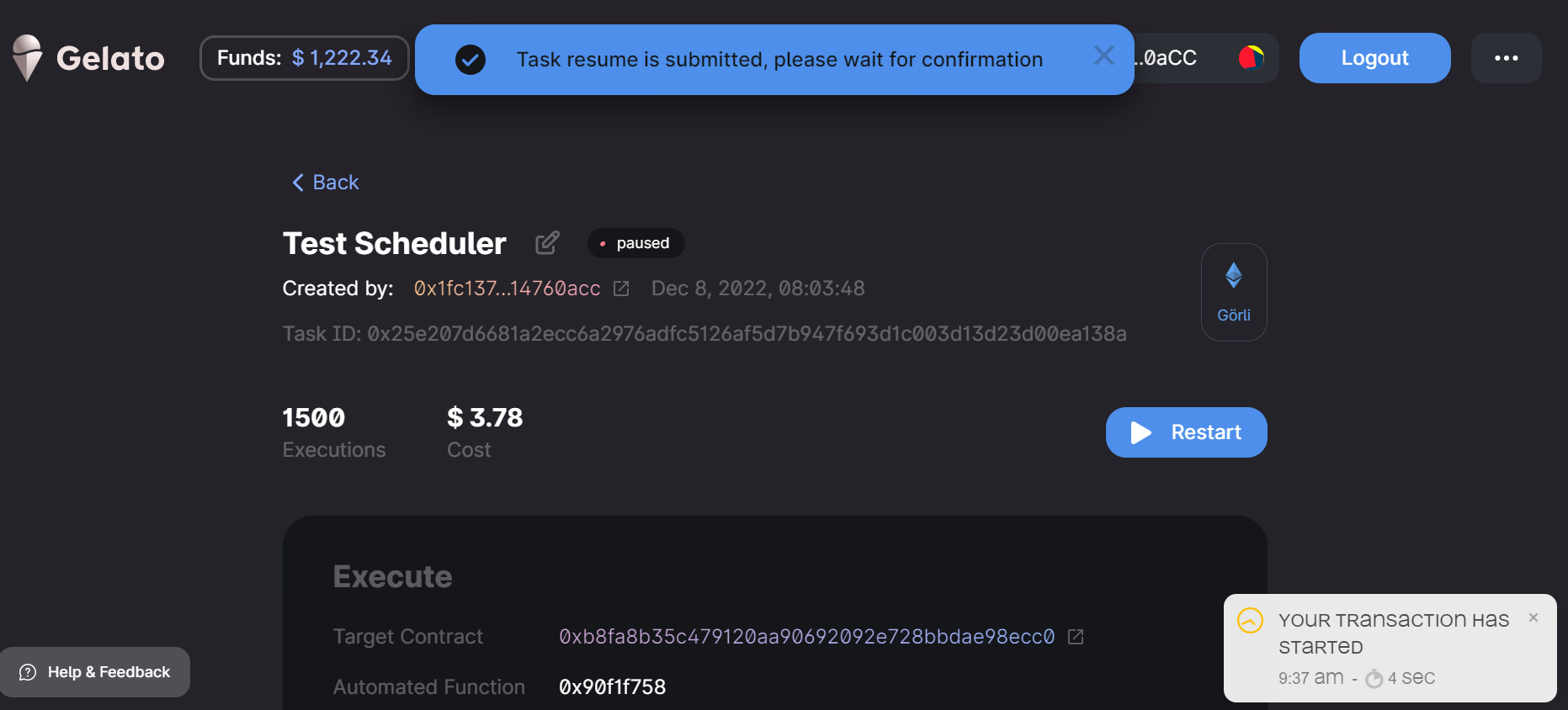
   .catch(error => {

     console.error(error);

     process.exit(1);

   });

* + Start running the Gelato Automation Smart Contract from alarm.sol smart contract with the same steps as listed previously.



* + Write the clientinformation.js file

const clientDataInput = require('./clientDataInput.js');

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

// For Hardhat

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

// 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 storebesu() {

    while (true) {

    const data = clientDataInput.input();

    // Wait for all of the Client Information to be stored

    const txc = await client.update(data.emailAddress, data.policy, data.duration, data.date\_of\_policy, data.phone, data.name, data.price);

    const receipt = await txc.wait()

      console.log(receipt.events);

      }

    }

    storebesu();

* + Write the payment.js file

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

// For Hardhat

const contract = require("../artifacts/contracts/mapping.sol/Mapping.json");

const contract1 = require("../artifacts/contracts/clientfactory.sol/PolicySmartContract.json");

// Provider

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

// Signer

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

// Contract

const payment = new ethers.Contract("0x7B3BEA60bD856E8095f73d7F79E5f932f98F2B35", contract.abi, signer);

async function paymentrpa() {

    while (true) {

      // View Data

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

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

      const viewId = await payment.getIdview(policy);

      const newClient = new ethers.Contract(viewId, contract1.abi, signer);

      const price = await newClient.getPrice();

      const premiumPayment = prompt("Please make payment for your Premium costing "+price+" per annum:$")

      const eventId = await payment.getId(policy);

      const receipt = await eventId.wait();

      console.log("Smart Contract Address for Policy Id", policy,"is", viewId );

      console.log(receipt.events);

    }

  }

  paymentrpa();

* + Write the rpaDaily.js file (with all the required integrations and configurations you have previously created)

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

const sgMail = require('@sendgrid/mail');

const API\_KEY =

'SG.nVF3DZJbTK6O1x9abcByDw.q5eDd\_\_vSQspRaTYg8tqpgVJ-vWfNJMVUFdeGPLIicA';

sgMail.setApiKey(API\_KEY)

const date = require('date-and-time');

const fs = require('fs');

const { initializeApp } = require('firebase/app');

const { getFirestore, collection, getDocs, onSnapshot,

  addDoc, deleteDoc, doc,

  query, where, querySnapshot, setDoc } = require('firebase/firestore');

// For Hardhat

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

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

// Provider

const alchemyProvider = new ethers.providers.AlchemyProvider(network="goerli", "jJopiviW6VYA5ngL2Z5gX45xhXwVTnRI");

// Signer

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

// Contract

const AlarmContract = new ethers.Contract("0xb8FA8b35c479120Aa90692092E728bBdAe98EcC0", contract.abi, signer);

// For Hardhat

const contract1 = require("../artifacts/contracts/clientFactory.sol/SmartContractFactory.json");

const contract2 = require("../artifacts/contracts/mapping.sol/Mapping.json");

const contract3 = require("../artifacts/contracts/clientfactory.sol/PolicySmartContract.json")

// Provider

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

// Signer

const signer1 = new ethers.Wallet("06b91f40bafe25bb97844ca576675d475ad4c197a4acab0f00a7dabf8326d480", provider);

// Contract

const Client = new ethers.Contract("0x3f21BC64076e7c9ed8695d053DCCBE6D8d5E6f43", contract1.abi, signer1);

const payment = new ethers.Contract("0x7B3BEA60bD856E8095f73d7F79E5f932f98F2B35", contract2.abi, signer1);

const firebaseConfig = {

  apiKey: "AIzaSyAITxJRyRKuAbbh0SeMfc\_WVSnfMorSITM",

  authDomain: "insurance-company-3bf30.firebaseapp.com",

  projectId: "insurance-company-3bf30",

  storageBucket: "insurance-company-3bf30.appspot.com",

  messagingSenderId: "686105572117",

  appId: "1:686105572117:web:f2b2fb8ea5a52d25a36f1c",

  measurementId: "G-TJ3EHNL58T"

};

  // init firebase app

  initializeApp(firebaseConfig)

  // init services

  const db = getFirestore()

async function testrpa() {

    abi = ["event Plus(uint plus)"];

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

    abi = ["event Map(address contractAddress"];

    console.log("Waiting for event");

    Client.on('Policy', async (smartContractAddr, email, policyId, duration, date\_of\_policy, phone, name, price) => {

      console.log("Policy Id:", policyId);

      console.log("Smart Contract Address:", smartContractAddr);

      const txc = await payment.setId(policyId, smartContractAddr);

      await txc.wait();

      const viewId = await payment.getIdview(policyId);

      console.log(viewId," has successfully been paired with", policyId);

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

      const policyDate = new Date(date\_of\_policy);

      const policyDate1 = date.format(policyDate, 'DDMM');

      const policyDate2 = date.format(policyDate, 'DD/MM/YYYY');

      console.log("Date of Policy:", policyDate2);

      console.log("DDMM =>", policyDate1);

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

      await setDoc(doc(db, '0', 'unpaid', policyDate1, smartContractAddr), {}) // Set to Unpaid

    })

    payment.on('Map', async (smartContractAddr) => {

      const newClient = new ethers.Contract(smartContractAddr, contract3.abi, signer1);

      const getPolicyDate = await newClient.getDate\_of\_policy();

      const policyDate = new Date(getPolicyDate);

      const policyDate1 = date.format(policyDate, 'DDMM');

      const policyDate2 = date.format(policyDate, 'DD/MM/YYYY');

      console.log("Date of Policy:", policyDate2);

      console.log("DDMM =>", policyDate1);

      console.log("Your payment has successfully been made!");

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

      await setDoc(doc(db, '1', 'paid', policyDate1, smartContractAddr), {}) // Add to Paid

      await deleteDoc(doc(db, '0', 'unpaid', policyDate1, smartContractAddr), {}) // Remove from Unpaid

    })

    let now = new Date('2022,12,23');

    const currenttime = date.format(now, 'DD/MM/YYYY');

    console.log("Current Time =", currenttime);

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

    AlarmContract.on('Plus', async (plus) => {

      if (plus != 25) {

        const updatedtime = date.addDays(now, 1);

        const then = new Date(updatedtime);

        now = then

        const updatedformattedtime = date.format(now, 'DDMM'); // Updated Time

        console.log("Updated Date =", updatedformattedtime);

        //now.setDate(now.getDate() - 1); // Minus 1 Day to the Date

        const oneMonthLater = date.addMonths(now, 1);

        const notification1 = date.format(oneMonthLater, 'DDMM'); // Buliding Notification 1

        console.log("1st Notification Time =", notification1); // 1 month later

        // Continue Building Notifications 2 - 4

        const twoWeeksLater = date.addDays(now, 14);

        const notification2 = date.format(twoWeeksLater, 'DDMM'); // Building Notification 2

        console.log("2nd Notification Time =", notification2); // 2 weeks later

        const threeDaysLater = date.addDays(now, 3);

        const notification3 = date.format(threeDaysLater, 'DDMM'); // Building Notification 3

        console.log("3rd Notification Time =", notification3); // 3 days later

        const sevenDaysBefore = new Date(now.setDate(now.getDate() - 3)); // Building the Overdue Notification

        const overdue = date.format(sevenDaysBefore, 'DDMM'); // 3 days before

        console.log("Overdue Notification Time =", overdue);

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

        // Add back 3 days to the Updated Time

        now.setDate(now.getDate() + 3);

        const colRef = collection(db, '1', 'paid', updatedformattedtime); // Final Check = Updated Time

        const subscribe = onSnapshot(colRef, async (querySnapshot) => {

            const contractAddress = [];

        querySnapshot.forEach((doc) => {

          contractAddress.push(doc.id);

          console.log(contractAddress);

        });

        for (index = 0; index < contractAddress.length; index++) {

            //await setDoc(doc(db, '0', 'unpaid', updatedformattedtime, contractAddress[index]), {}); // Final Check

            //await deleteDoc(doc(db, '1', "paid", updatedformattedtime, contractAddress[index]), {}); // Update Paid to Unpaid

            //console.log(contractAddress[index], "has been successfully reset from paid to unpaid");

            console.log("rpaDaily.js payment status has already been set to 0 by default for demo purposes");

        }});

        const colRefNoti1 = collection(db, '0', 'unpaid', notification1);

        const colRefNoti2 = collection(db, '0', 'unpaid', notification2);

        const colRefNoti3 = collection(db, '0', 'unpaid', notification3);

        const colRefOverdue = collection(db, '0', 'unpaid', overdue);

        const subscribe1 = onSnapshot(colRefNoti1, async (querySnapshot) => {

            const address1 = [];

          querySnapshot.forEach((doc) => {

            address1.push(doc.id);

          });

          for (index = 0; index < address1.length; index++) {

            console.log(address1[index]);

            const emailClient = new ethers.Contract(address1[index], contract3.abi, signer1);

            const name = await emailClient.getName();

            const price = await emailClient.getPrice();

            const message = {

                to: 'genshinimapact1997@gmail.com',

                // from: 'ishangill2003@gmail.com',

                from: {

                    name: 'Enterprise Insurance Agency',

                    email: 'ishangill2003@gmail.com'

                },

                subject: 'Insurance Premium Payment Reminder',

                body: 'Insurance Premium Payment Reminder',

                html: '<h4>Dear '+name+'<h4/> <br><br> <p>This is your 1st Notification Reminder to inform you that you have yet to pay your insurance premium costing '+price+' per annum<p/>'

            };

            sgMail

            .send(message)

            .then((response) => console.log('Email sent...'))

            .catch((error) => console.log(error.message));

          }

        });

        const subscribe2 = onSnapshot(colRefNoti2, async (querySnapshot) => {

          const address2 = [];

          querySnapshot.forEach((doc) => {

            address2.push(doc.id);

          });

          for (index = 0; index < address2.length; index++) {

            console.log(address2[index]);

            const emailClient = new ethers.Contract(address2[index], contract3.abi, signer1);

            const name = await emailClient.getName();

            const price = await emailClient.getPrice();

            const message = {

                to: 'genshinimapact1997@gmail.com',

                // from: 'ishangill2003@gmail.com',

                from: {

                    name: 'Enterprise Insurance Agency',

                    email: 'ishangill2003@gmail.com'

                },

                subject: 'Insurance Premium Payment Reminder',

                body: 'Insurance Premium Payment Reminder',

                html: '<h4>Dear '+name+'<h4/> <br><br> <p>This is your 2nd Notification Reminder to inform you that you have yet to pay your insurance premium costing '+price+' per annum<p/>'

            };

            sgMail

            .send(message)

            .then((response) => console.log('Email sent...'))

            .catch((error) => console.log(error.message));

          }

        });

        const subscribe3 = onSnapshot(colRefNoti3, async (querySnapshot) => {

          const address3 = [];

          querySnapshot.forEach((doc) => {

            address3.push(doc.id);

          });

          for (index = 0; index < address3.length; index++) {

            console.log(address3[index]);

            const emailClient = new ethers.Contract(address3[index], contract3.abi, signer1);

            const name = await emailClient.getName();

            const price = await emailClient.getPrice();

            const message = {

                to: 'genshinimapact1997@gmail.com',

                // from: 'ishangill2003@gmail.com',

                from: {

                    name: 'Enterprise Insurance Agency',

                    email: 'ishangill2003@gmail.com'

                },

                subject: 'Insurance Premium Payment Reminder',

                body: 'Insurance Premium Payment Reminder',

                html: '<h4>Dear '+name+'<h4/> <br><br> <p>This is your Final Notification Reminder to inform you that you have yet to pay your insurance premium costing '+price+' per annum<p/>'

            };

            sgMail

            .send(message)

            .then((response) => console.log('Email sent...'))

            .catch((error) => console.log(error.message));

          }

        });

        const subscribe4 = onSnapshot(colRefOverdue, async (querySnapshot) => {

          const address4 = [];

          querySnapshot.forEach((doc) => {

            address4.push(doc.id);

          });

          for (index = 0; index < address4.length; index++) {

            console.log(address4[index]);

            const emailClient = new ethers.Contract(address4[index], contract3.abi, signer1);

            const name = await emailClient.getName();

            const price = await emailClient.getPrice();

            const message = {

                to: 'genshinimapact1997@gmail.com',

                // from: 'ishangill2003@gmail.com',

                from: {

                    name: 'Enterprise Insurance Agency',

                    email: 'ishangill2003@gmail.com'

                },

                subject: 'Insurance Premium Payment Reminder',

                body: 'Insurance Premium Payment Reminder',

                html: '<h4>Dear '+name+'<h4/> <br><br> <p> Your Insurance premium costing '+price+' per annum has been terminated since payment has not been made! Do Contact our Staff if you are keen on renewing your Insurance Premium.<p/>'

            };

            sgMail

            .send(message)

            .then((response) => console.log('Email sent...'))

            .catch((error) => console.log(error.message));

              await setDoc(doc(db, '-1', 'overdue', overdue, address4[index]), {}); // Overdue Check

              await deleteDoc(doc(db, '0', "unpaid", overdue, address4[index]), {}); // Update Unpaid to Overdue

              console.log(address4[index], "has been successfully reset from unpaid to overdue");

          }

        });

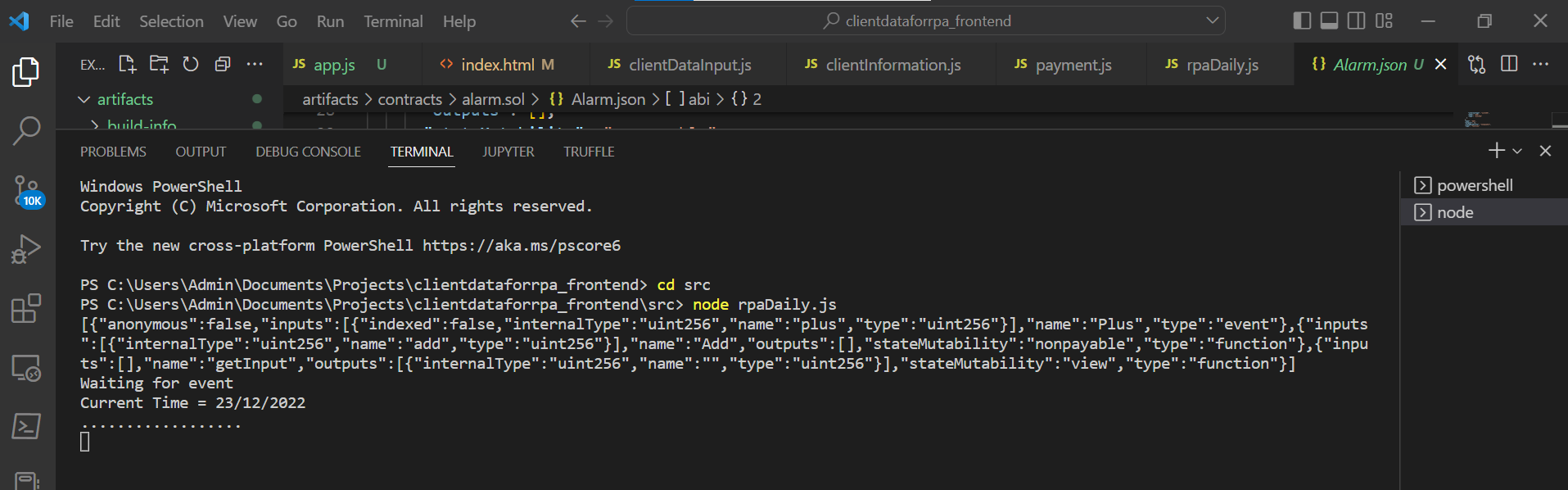
      };

    });

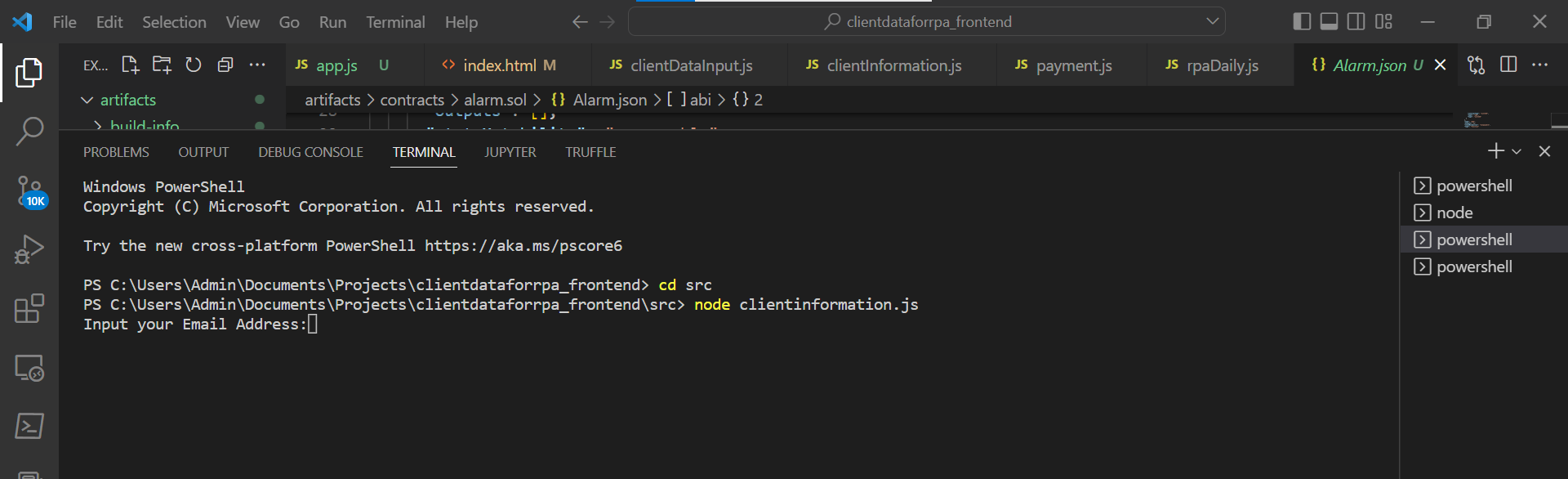
}

  testrpa();

* Run the Following Backend Files in the following order
  + Run rpaDaily.js with the following commands



* + Run clientinformation.js and payment.js with the same format



Congrats! You have successfully completed setting up the Private Insurance Company Automation Backend Files!