Energy Meters: Studio e definizione dell'architettura di un sistema informativo per l'analisi e la correlazione di dati estratti da database sui consumi energetici ad uso domestico

Attività svolta nell'ambito dell'Avviso promosso dal Ministero dell'Università e della Ricerca per la presentazione di Idee progettuali per Smart Cities and Communities and Social Innovation di cui al D.D. n. 391/Ric. del 5 luglio 2012 e ss.mm.ii..
SIN_00968 THE LEARNING METERS NETWORK:
workpackage formativo del SCN_00398
CUP J49G14000140008

Marco Crotta

UNIVERSITÀ DEGLI STUDI DI PERUGIA
Dipartimento di Ingegneria
D.D. n. 110/2020

Indice

Introduzione

Il progetto "Energy Meters" presenta un sistema informativo per l'analisi e la correlazione di dati basati su database relativi ai consumi energetici ad uso domestico. Scopo del progetto è definire una struttura che sia atta a raccogliere, immagazzinare, conservare e rendere disponibili i dati di consumo di una comunità energetica (definita più avanti).

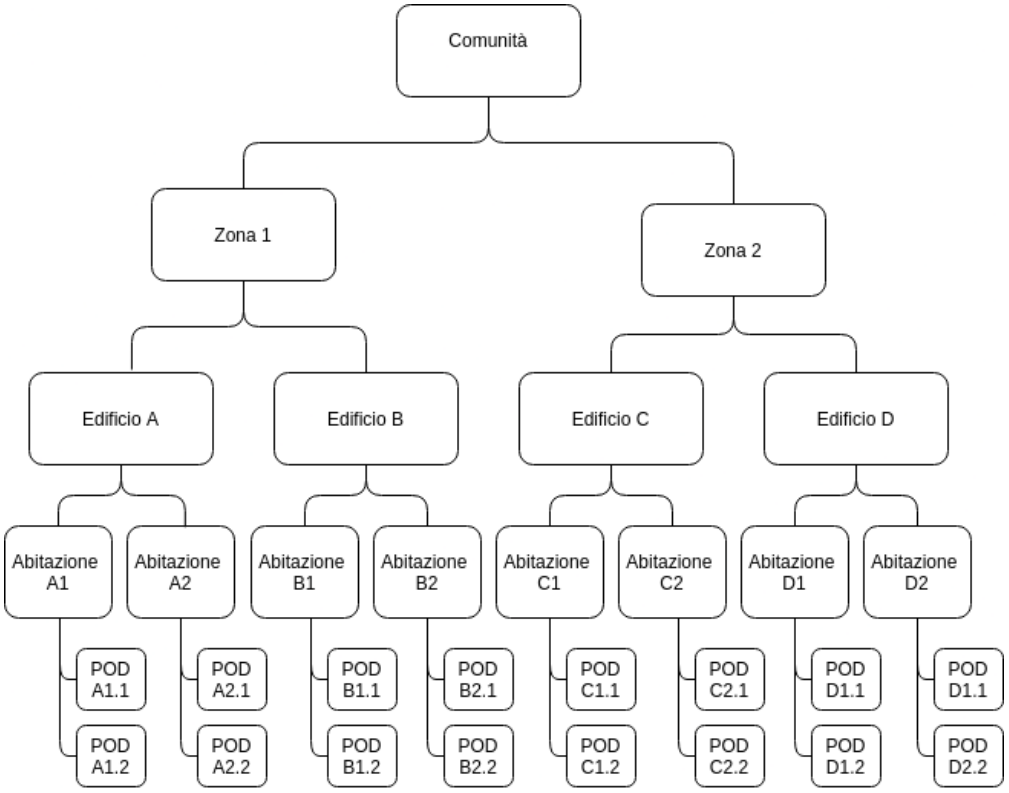
Data la natura del progetto, dei dati trattati e delle possibili successive applicazioni, particolare attenzione è stata messa sulla conservazione del dato e sulla possibilità di sfruttare lo stesso e le elaborazioni che ne derivano in contesti in cui questi possano avere valore economico, contrattuale o legale.

Panoramica della soluzione

Scenario

La realtà osservata è composta da diversi elementi. Di seguito un elenco degli stessi e la relazione gerarchica e di appartenenza tra questi:

- **comunità energetica:** un insieme di zone o aree geografiche da cui la comunità è composta
- **zone**: aree geograficamente contigue ed identificabili (terreni, frazioni, isolati...)
- **edifici:** trattandosi di consumi domestici/residenziali, vengono identificati i singoli edifici in cui hanno luogo i consumi
- **abitazioni:** (casa privata, negozio, sede aziendale)
- **pod:** un centro di raccolta dati provenienti dai sensori
- **sensore:** strumento di misurazione collocato suei punti di prelievo dell'energia all'interno dell'abitazione (elettrodomestici, prese di corrente...)



Un POD può essere dotato di uno o più sensori atti a effettuare diverse misurazioni. I consumi saranno rilevati direttamente dai sensori dei **POD**. Questi sono **device IOT** collegati alla rete in grado di misurare i dati di consumo ed ambientali. Il **POD** potrà essere in grado di richiedere una registrazione di una misurazione ad un intermediario o, preferibilmente, di effettuare autonomamente la registrazione della misurazione in modo sicuro.

Utenze, competenze, responsabilità

Da un punto di vista funzionale dovranno essere considerati diversi attori che operano su una realtà come quella descritta. Considereremo questi attori come utenti del sistema. Al fine di attribuire, separare e gestire le diverse competenze e responsabilità, gli attori saranno divisi in categorie. In particolare avremo:

- **utenze amministrative o privilegiate:** si tratta di utenze che hanno accesso alle parti di configurazione, impostazione e gestione della piattaforma. Saranno utilizzate diverse utente amministrative responsabili di diverse parti della piattaforma. In particolare gli amministratori avranno la possibilità di regolare l'accesso delle utenze non privilegiate, finali o di servizio alle parti di loro competenza
- **utenze finali o non-privilegiate**: si tratta di utenze gestite direttamente dai proprietari, consumatori o residenti della comunità energetica che materialmente sono responsabili dei consumi effettuati
- **utenze di servizio:** si tratta di utenze utilizzate da programmi, script o altri software per accedere alle diverse parti della soluzione per ricevere, gestire e amministrare i dati.

Utilizzo della Blockchain

Dalle premesse fatte finora, la prima scelta architetturale fondamentale fatta, è stata quella di non appoggiarsi su un generico database, ma sulla migliore soluzione attualmente disponibile per quello che riguarda la sicurezza, l'affidabilità e la verificabilità del dati, ovvero la **Blockchain**. Le caratteristiche di inalterabilità del dato, di sicurezza, date dalle primitive crittografiche e dalle firme digitali, e l'opponibilità a terzi del dato contenuto (vedi l'articolo 8-ter del Testo Coordinato del Decreto-Legge 14 dicembre 2018, n. 135, Appendice A) rendono gli smart contract lo strumento migliore e più economico per portare queste garanzie.

I dati quindi saranno dapprima letti dai sensori, veicolati dai pod, quindi ricevuti dagli smart contract della piattaforma e scritti in blockchain tramite degli smart contract. I dati depositati in blockchain potranno essere successivamente letti ed elaborati secondo quanto necessario.

Apparentemente questo passaggio (mettere i dati in blockchain e poi successivamente prelevarli da li) potrebbe erroneamente sembrare un passaggio accessorio poco sensato. Il senso di questo passaggio è di lasciare traccia perenne ed inalterabile del funzionamento dello smart contract e dei valori letti così che nessuna diatriba o dubbio possa mai sollevarsi rispetto alla completezza e correttezza degli stessi.

La scelta di adoperare la blockchain innesca automaticamente la necessità di dover precisare quale blockchain utilizzare nello specifico. Fare questa scelta richiede di considerare diversi aspetti e il loro impatto sul lungo termine. Tra questi aspetti abbiamo:

1. tipologia o classe di blockchain
2. interoperabilità ed adozione di standard

3. velocità della rete
4. possibilità di migrazione della soluzione
5. costi relativi all'esercizio per le transazioni
6. costi di manutenzione di un nodo e/o della struttura

Riguardo ai punti 1 e 6: per la natura dell'applicazione e il tipo di soluzione che si vuole implementare impone indiscutibilmente di utilizzare una **blockchain pubblica** (essendo che le cosiddette blockchain private non danno garanzia alcuna di immutabilità del dato e quindi verrebbe a cadere la possibilità di opponibilità a terzi del dato registrato). La scelta di utilizzare una blockchain pubblica elimina la necessità di realizzare, mantenere e quindi pagare una propria soluzione ospitata su propria struttura, abbattendo i costi di manutenzione (CTO), quelli relativi all'acquisto di hardware specializzato, infrastruttura di rete, personale.
Un'altro vantaggio derivante dall'utilizzo della blockchain pubblica è quello di avere già a disposizione diversi **strumenti e servizi** a disposizione (quali strumenti di monitoraggio, verifica ed explorer), gratuiti, testati, che essendo gestiti e mantenuti da terze parti del tutto estranee al progetto sono un'ulteriore garanzia della trasparenza e verificabilità della soluzione.

Riguardo al punto 2 e 4: si vuole una soluzione che aderisca a degli standard diffusi per cui la stessa soluzione che andremo a definire sia implementabile senza modifiche sostanziali su diverse reti blockchain. In questo senso la soluzione più indicata è una blockchain basata su **Ethereum Virtual Machine (EVM)** che permetta di sviluppare smart contract in linguaggio **solidity**.

Riguardo al punto 5: è necessario che il costo di operazione della blockchain sia il più basso possibile e che possa e si mantenga quanto più possibile costante nel tempo, diversamente la variazione del costo potrebbe essere un elemento in grado di mettere in discussione l'intero progetto fino a rendere necessaria l'intera migrazione dello stesso o il suo arresto.

Riguardo al punto 3: ipotizzando (come da documentazione ricevuta e successivi chiarimenti) che ogni sensore di ogni pod registri un dato all'ora, avendo presumibilmente massimo 12 sensori per pod ed avendo un numero di 500 pod in una comunità, otteniamo che potrebbe essere necessario fare 144.000 transazioni al giorno, equivalenti ad una media di 1,7 tx/s con punte massime di 50 al secondo. Va tenuto poi in considerazione che, qualora il picco di richieste non fosse immediatamente risolto, le richieste formerebbero una coda che verrebbe smaltita gradualmente nel tempo.

Per tutte le considerazioni di cui sopra, in fase di avvio la scelta è ricaduta sulla blockchain **Quadrans** (https://quadrans.io/). Quadrans è una piattaforma blockchain pubblica, open source, basata su EVM e totalmente compatibile con Ethereum, nata in italia, a basso impatto ambientale, dai consumi ridotti, dai costi estremamente bassi e senza variazioni sensibili dei costi vivi, studiata appositamente ed esclusivamente per applicazioni di tipo aziendale ed industriale o per la ricerca. Ha un tasso di transazioni al secondo pari a circa 80 tx/s più che idoneo a gestire i numeri in gioco.

Il resto del documento farà riferimento a questa blockchain. Si rimanda ai siti di Quadrans e della Fondazione Quadrans per approfondimenti e documentazione dedicata (https://docs.quadrans.io/)

Impiego degli Smart Contracts

Gli smart contract sono script software depositati in blockchain che funzionano integrandosi a quel ecosistema. A scanso di equivoci, contrariamente a quanto spesso diffuso, si precisa che non si tratta in alcun modo di contratti e che non sono automatici. Essendo in blockchain gli smart contract godono direttamente di tutte le proprietà tipiche della transazioni dei una blockchain. Sono pertanto immutabili, distribuiti e replicati su ogni nodo della blockchain e la loro esecuzione è distribuita, non può quindi essere interrotta o impedita per nessun modo e non si può alterarne il risultato. Inoltre le caratteristiche della blockchain fanno si che l'esecuzione dello smart contract avverrà in parallelo su tutti i nodi della rete, partirà con gli stessi input e dovrà deterministicamente concludersi con gli stessi risultati su tutti i nodi.

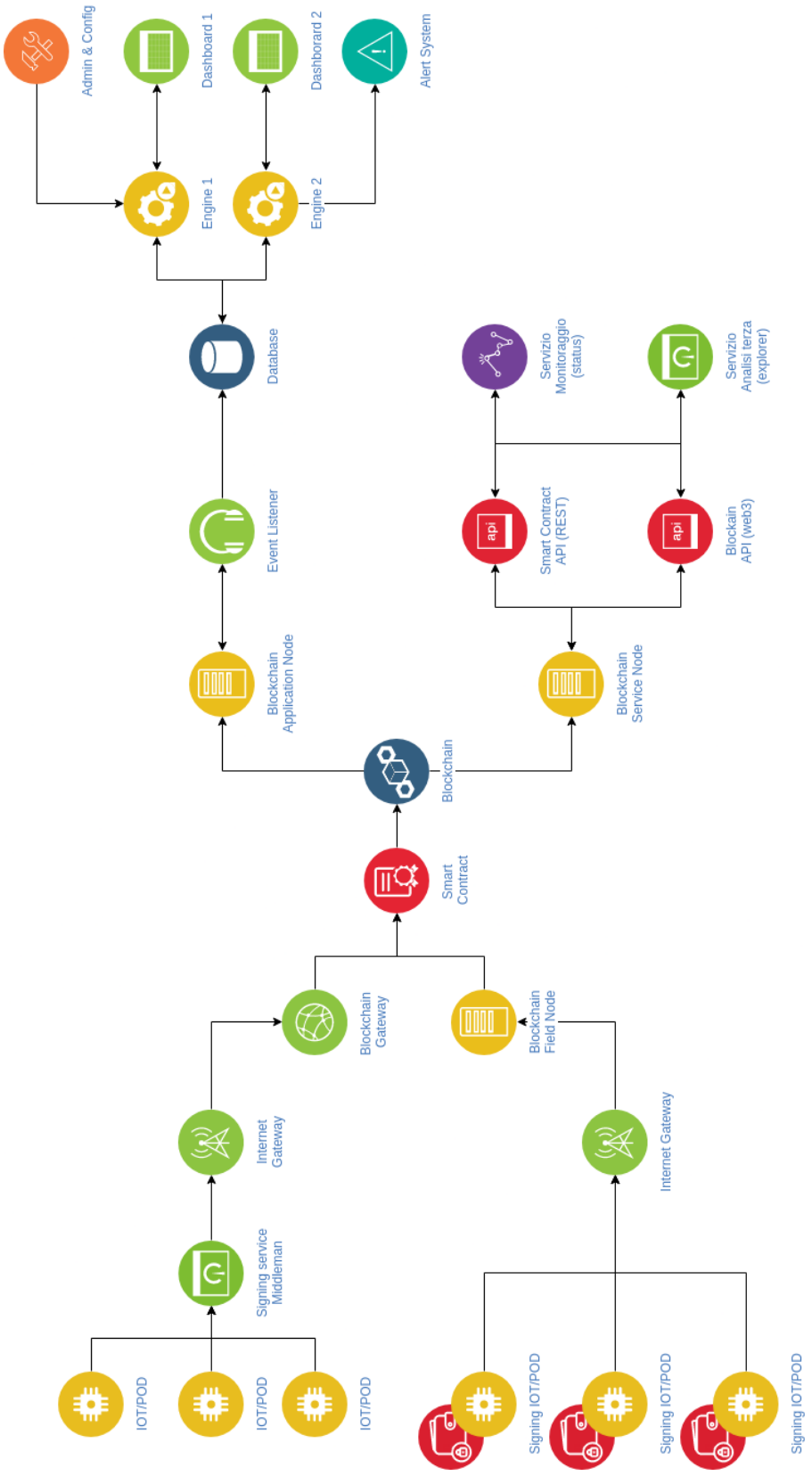
Gli smart contract, essendo dei software, possono avere una struttura dati interna. Anche questa struttura dati dovrà essere identica per ogni istanza dello smart contract su qualsiasi nodo. Non si può verificare che sulla stessa blockchain esistano due nodi che attribuiscono valori diversi per i dati di uno stesso smart contract. Inoltre, appoggiandosi sulla blockchain, questi software possono essere azionati solo mediante delle transazioni, e questo fa sì che in automatico l'autore di una transazione indirizzata allo smart contract sia necessariamente dotato di un suo indirizzo e/o wallet per immettere in rete le transazioni firmate digitalmente. Questo implica che ogni modifica di stato o aggiornamento ai dati dello smart contract sia bi-univocamente associabile ad un address di un utente che dispone una transazione firmata digitalmente ed incorruttibile.

Nel nostro contesto gli smart contract verranno quindi usati per:

- contenere i dati relativi agli utenti abilitati
- contenere i dati relativi ai POD abilitati
- modificare i dati relativi a utenti e POD
- mantenere una relazione di appartenenza tra utenti e POD
- ricevere i messaggi di utenti e POD
- applicare le regole di validazione ai messaggi
- depositare in blockchain in modo sicuro i dati ricevuti e validati

Nei capitoli successivo vedremo in dettaglio la realizzazione degli smart contract.

Diagramma della Soluzione



Spiegazione del diagramma

Con riferimento al diagramma strutturale precedente, possiamo descrivere la soluzione proposta come segue.

Raccolta dati ed firma delle transazioni:

- I dati sono raccolti da device **IOT/POD** (prima colonna di elementi a sinistra).
- le device possono essere “semplici” (parte superiore) o integrare un wallet.
- le device semplici non sono in grado di fare operazioni complesse, quindi si limiteranno a raccogliere i dati ed esporre un pacchetto dati.
- le device avanzate sono dette “**signing IOT/POD**” in quanto hanno capacità di calcolo avanzate e quindi la possibilità di fornire i dati all’interno di una transazione per la blockchain completa di firma digitale.
- le device semplici devono passare i dati ad un intermediario o **signing service middleman** che ha la responsabilità di raccogliere i dati e formare le transazioni da inviare in blockchain.
- le signing device hanno solo la necessità di potersi connettere ad un **internet gateway** o ad una connessione di rete tramite cui veicolare la transazione.
- le signing device potranno inviare transazioni direttamente alla blockchain o esporre un set di dati firmato digitalmente.
- in alternativa o come misura di emergenza, i dati possono essere inseriti manualmente dagli stessi utenti.

Invio delle transazioni allo smart contract:
Le transazioni potranno essere inviate alla blockchain in due modi diversi:

- il primo modo consiste nel collegarsi direttamente ad un nodo della blockchain sul campo o **blockchain field node**
- il secondo modo consiste nel veicolare le transazioni mediante un **blockchain gateway** (ex Infura, Moralis, rpc.quadrans.io)

Smart contract e scrittura in Blockchain:

- la transazione quindi sarà minata ed elaborata dallo **smart contract** che metterà i dati in **blockchain**

Lettura dei dati:

- I dati in blockchain potranno essere letti da qualsiasi nodo. Possiamo ipotizzare di avere nodi blockchain dedicati all’applicazione o **blockchain application node**, e nodi dedicati ad altri servizi o **blockchain service node**
- i blockchain service node potranno essere interrogati, sia a livello di blockchain che a livello di smart contract, tramite **API Web3.js e/o REST** per raccogliere dati quali l’andamento della blockchain e delle transazioni all’interno
- i dati raccolti dalle API saranno utilizzabili per lo sviluppo di strumenti accessori di **servizi di analisi (explorer)** e/o di **servizi di monitoraggio (status)**
- il **blockchain application node** (di cui possono esistere diverse istanze) sarà interrogato da un **event listener**, ovvero un software che sarà in ascolto degli **eventi** (log) emessi dallo **smart contract**

Raccolta dati:

- gli eventi saranno depositati in un **database** tradizionale, in modo da raccogliere tutte le informazioni e le transazioni cui si riferiscono per referenza futura. Si noti che i dati restano comunque in blockchain e sono quelli a fare fede, mentre la loro raccolta nel database è da considerarsi ai soli fini di comodità e velocità di elaborazione
- il **database** viene interrogato dagli **engine** di elaborazione che raccolgono i dati contenuti ed effettuano i calcoli e le aggregazioni del caso.
- gli engine saranno gestiti dagli utenti amministratori che potranno, ad esempio, impostare le configurazioni ed i parametri operativi degli engine.
- gli engine presenteranno i dati in **dashboard** dedicate, logs o report
- un alert system può essere utilizzato per inviare allarmi e notifiche qualora si verifichino anomalie e malfunzionamenti

Inserimento manuale dei dati

La soluzione prevede la possibilità di inserire manualmente i dati in caso di emergenza o per correggere manualmente disallineamenti o ritardi dovuti a guasti dei POD. Si noti che questa soluzione potrebbe apparentemente rappresentare un punto di debolezza per la soluzione ma non è così. Le transazioni manuali sono possibili solo per gli utenti abilitati, solo se fatte con l'indirizzo registrato dell'utente, e solo utilizzano un metodo di registrazione dedicato.
In altre parole sarà sempre possibile distinguere tra un dato inserito da un POD, da un signing service o da un utente, e sapere con precisione l'autore della specifica transazione. Questo consente di effettuare controlli e verifiche sia in tempo reale che in momenti successivi per appurare il corretto uso della piattaforma ed identificare ed escludere utilizzi non corretti e comportamenti opportunistici.

Punti di attenzione sulle device

Particolare attenzione va posta sulla scelta delle tipologie di device IOT/POD impiegate. Nello schema precedente sono state introdotte anche le device di tipo semplice per completezza e per dimostrare la flessibilità di questo tipo di soluzioni, tuttavia queste device sono sconsigliate.
Il motivo è che il dato emesso da queste device potrebbe non essere protetto in modo adeguato. Il dato infatti viene emesso dalle device semplici in chiaro e senza firma crittografica. Attraversa una rete potenzialmente insicura, per cui il dato potrebbe essere alterato prima di raggiungere lo smart contract. Ad esempio il signing middle man potrebbe, se manomesso, alterare i valori prima di formare la transazione ed apporre la firma digitale.

Diversamente il signing IOT/POD sono in grado di emettere da soli una transazione firmata o un pacchetto dati firmato, che risulta essere inalterabile alla fonte e lungo tutto il percors fino allo smart contract. Il dato emesso dalla device è sicuro solo dopo la firma digitale che lo rende inalterabile. Una volta che il dato è reso inalterabile, l'unico attacco che si può fare alla soluzione è quello di impedire la registrazione del dato interrompendo il traffico di rete. Tuttavia questa operazione lascerebbe dei vuoti e disallineamenti temporali evidenti.

Si noti inoltre che le signing IOT/POD sono rappresentate nello schema e con un wallet associato. Questo vuol dire che la device ha al suo interno la chiave privata necessaria per firmare le transazioni. Si hanno quindi due possibilità:

- la device emette transazioni blockchain complete, dovrà quindi avere un saldo associato al suo address con cui pagare autonomamente le transazioni.
- la device emette pacchetti dati firmati che saranno trasformati in una transazione da un intermediario che pagherà la transazione al posto della device

I due casi permettono di gestire in modo diverso le risorse criptoeconomiche della blockchain a seconda delle necessità. Si noti che in qualunque caso il dato emesso dalla device non è alterabile e si ha totale visibilità su quali siano le modalità di arrivo dei dati nella blockchain e quali siano gli attori responsabili.

In particolare la seconda modalità fa uso dei messaggi a firma digitale e consente di avere un solo wallet di cui controllare periodicamente il saldo in criptovaluta con un notevole beneficio in termini di gestione.

Smart Contract

Tutta la parte di registrazione dei dati in blockchain è affidata ad uno smart contract dedicato, lo smart contract "*ConsumptionTracker.sol*" che ha la responsabilità di :

- contenere gli address degli utenti abilitati
- contenere i dati relativi ai POD abilitati
- mantenere la relazione di appartenenza tra utenti e POD
- modificare i dati relativi a utenti e POD
- ricevere le richieste di registrazione da utenti e POD
- applicare le regole di validazione ai messaggi
- depositare in blockchain in modo sicuro i dati ricevuti e validati

Entità definite

Utente o owner

Ogni utente o owner è identificato da due address, ognuno dei quali nella forma di stringa esadecimale da 160 bit (20 bytes).

Ex: 0xee1ef5499ef77183309423ecee907861ecf8125c

- il primo è l'indirizzo che viene usato per firmare le transazioni delle registrazioni,
- il secondo è l'indirizzo di un wallet privato dell'utente che potrebbe essere impiegato in successive implementazioni.

Questa separazione è introdotta per consentire una maggiore flessibilità sulle successive implementazioni della piattaforma. A titolo di esempio si descrive un possibile utilizzo in uno scenario:

- il primo indirizzo, per la firma delle transazioni, potrebbe essere gestito direttamente dall'utente, dal POD, o da un intermediario. Il rapporto deve essere 1:1
- il secondo indirizzo potrebbe essere usato per la contabilizzazione, pagamenti, ricompense derivanti dall'utilizzo della piattaforma. E' in teoria possibile che diversi utenti/POD utilizzino lo stesso wallet in alcuni contesti

Esiste un solo **utente privilegiato** o **SuperUser** che è l'owner dello smart contract che ha la possibilità di abilitare e disabilitare gli altri utenti.
Ogni utente può essere proprietario di più POD. Deve essere l'utente stesso ad inserire i dati dei POD che dichiara essere propri. Sempre l'utente potrà disconoscere un POD precedentemente dichiarato proprio (ex, guasto e sostituzione).

POD

Ogni POD è identificato da un ID che consiste in un numero intero positivo a 256bit (32byte). Il numero può essere rappresentato sia in forma decimale che esadecimale.

Ex: 1020569450

Ex: 0x876a0a47d78363f2c814d65612c28d

Ogni POD deve avere un identificativo diverso.
Il POD a sua volta avrà anche un address, necessario per la validazione delle transazioni firmate dal POD stesso. E' possibile, ma fortemente sconsigliato, che due POD con ID necessariamente differente possano usare lo stesso address.

Sensore

I sensori sono associati ad un POD. Non è necessario che il POD definisca o dichiari l'identità dei sensori. Quando il POD effettua una registrazione dai dichiarerà l'identificativo del sensore. Idealmente ogni sensore avrà un identificativo hardware univoco simile ad un MAC-Address di un dispositivo etherenet, e viene rappresentato da un numero 8byte in esadecimale

Ex: 0xab60483f81628d2e

Misurazione e registrazione

Le misurazioni dei consumi della comunità energetica sono basate su tre funzioni dedicate:

- *recordFromPod*: registra i dati inviati tramite transazione da una signing-device
- *recordFromOwner*: registra i dati inviati tramite transazione da un utente
- *recordFromSigner*: registra i dati firmati digitalmente inviati tramite un intermediario

Modalità di utilizzo

Di seguito una serie di passi che descrivono l'utilizzo dello smart contract:

Fase di setup
- lo smart contract è deployato dall'utente **S** che diventa **SuperUser** dello smart contract
- l'utente **S** abilita all'utilizzo dello smart contract gli utenti **A, B, C** fornendo per ognuno due indirizzi (vedi dopo)
- l'utente **A** dichiara di possedere 2 POD, **a1** e **a2**, inserendo gli identificativi numerici e gli address
- similmente **B** inserisce i dati per **b1** e **b2**
- **C** inserisce il pod **c1**, **c2**, **c3**
- l'utenza **D** prova ad inserire il pod d1 ma viene bloccato in quanto non abilitato da **S**
- l'utente **C** elimina il POD **c2** inserito per errore
- l'utente **S** elimina l'utente **C** inserito per errore. I POD **c1** e **c3** sono cancellati.

Fase di utilizzo:
- il POD **a1** entra in funzione e registra autonomamente i dati pagando le transazioni tramite la funzione *recordFromPod*

- il POD **b1** entra in funzione ed invia dati firmati all'utente **B**.
- l'utente **B** raccoglie i dati ricevuti dal POD **b1** e li invia allo smart contract tramite la funzione *recordFromSigner*
- l'utente **B** si accorge che il POD **b2** non funziona, ricalcola a mano i dati da inserire e aggiorna i dati mancanti tramite la funzione *recordFromOwner*

Funzioni implementate

Di seguito la lista dei metodi implementati dall' smart contract *ConsumptionTraker* corredata di input, output, visibilità e descrizione del funzionamento.
I parametri di input ed output saranno descritti come lista di coppie *<tipo nome_variabile>* separate da virgola. I tipi di dati si rifanno alla sintassi del linguaggio Solidity. Per quanto concerne la visibilità, i metodi contrassegnati come "*onlyOwner*" saranno eseguibili solo dall'utete il cui indirizzo corrisponde a quello della variabile *smartContractOwner*, che viene impostata una sola volta in fase di deploy con l'indirizzo del proprietario dello smart contract.

funzione: enableAddress
input: address _who, address _wallet
output: -
visibilità: public onlyOwner
funzionamento: l'utente SuperUser è l'unico che può utilizzare questa funzione, tutti gli altri utenti che provassero a richiamare la funzione vedrebbero la transazione andare in revert e fallire. Il SuperUser presenta una coppia di indirizzi per l'utente: il primo è l'indirizzo per la firma delle transazioni, il secondo è il wallet deputato ad altri impieghi finali della piattaforma. L'indirizzo passato sarà marcato come abilitato. La corretta terminazione della transazione a buon fine emette un evento *"Enabled"*

funzione: disableAddress
input:address _who
output: -
visibilità: public onlyOwner
funzionamento: l'utente SuperUser è l'unico che può utilizzare questa funzione, tutti gli altri utenti che provassero a richiamare la funzione vedrebbero la transazione andare in revert e fallire. Il SuperUser presenta l'indirizzo principale dell'utente (il primo, l'indirizzo per la firma delle transazioni).
L'indirizzo passato sarà marcato come disabilitato. La corretta terminazione della transazione a buon fine emette un evento *"Disabled"*

funzione: adoptPod
input:uint _pod, address _podaddress
output: -
visibilità: public

funzionamento: la funzione è richiamabile solo dagli utenti il cui address è stato abilitato tramite la funzione enableAddress, tutti gli altri utenti che provassero a richiamare la funzione vedrebbero la transazione andare in revert e fallire. L'utente dichiara di essere proprietario del POD identificato con quel ID numerico e quell'indirizzo per la firma delle sue transazioni. La transazione fallisce se l'identificativo è già stato precedentemente usato. La corretta terminazione della transazione a buon fine emette un evento "Adopted".

funzione: rejectPod
input: uint _pod
output: -
visibilità: public
funzionamento: la funzione è richiamabile solo dagli utenti il cui address è stato abilitato tramite la funzione enableAddress, tutti gli altri utenti che provassero a richiamare la funzione vedrebbero la transazione andare in revert e fallire. L'utente dichiara di alienare il POD identificato con quel ID numerico. La transazione fallisce se l'identificativo non è già stato precedentemente indicato come adottato da parte di quello stesso utente. La corretta terminazione della transazione a buon fine emette un evento "Rejected".

funzione: recover
input: bytes32 _messagehash, bytes memory _signature
output: address
visibilità: internal pure
funzionamento: la funziona è interna e quindi non può essere richiamata da un utente, è accessibile solo da altre funzioni interne dello smart contract. Prende in input una hash di un messaggio ed una firma digitale. La funzione controlla che la firma digitale combaci con l'hash del messaggio. Se questa combacia legge la firma e restituisce l'address che ha effettuato quella firma su quell'hash.
La funzione sarà richiamata da recordFromSigner al fine di controllare la correttezza dei dati ricevuti.

funzione: isValidNonce
input: uint _pod, uint _nonce
output: bool
visibilità: internal
funzionamento: la funziona è interna e quindi non può essere richiamata da un utente, è accessibile solo da altre funzioni interne dello smart contract.

preambolo: Ogni transazione firmata ricevuta dalle recordFromSigner deve essere protetta da un attacco noto come "replay attack". Questo attacco consiste nel riproporre più volte lo stesso messaggio così da ottenere più volte gli stessi effetti. Ad esempio un intermediario malevolo potrebbe annotare i parametri di una chiamata recordFromSigner che registra consumi bassi, e sostituire le successive chiamate che riportano consumi alti con quella precedente, più conveniente.
Per evitare che i messaggi firmati siano riproposti questi dovranno contenere un numero progressivo, detto nonce. Lo smart contract terrà un contatore aggiornato del nonce di ogni

POD/signer per cui non sarà possibile riproporre messaggi vecchi. Questo impone che ogni POD firmi transazioni nuove con un nonce che incrementa progressivamente vanificando l'attacco.

La funzione isValidNonce riceve in input il nonce e l'address dell'emittente. Controlla se quel nonce è valido per quell'address mittente e ritorna un valore vero o falso a seconda dei casi. Se il nonce è valido verrà aggiornato il contatore dello smart contract e i successivi messaggi da quell'address dovranno avere un valore maggiore.

funzione: record
input: address _podAddr, uint _podId, bytes8 _sensorId, int32[] memory _sensorData, uint _when, uint _mode
output: -
visibilità: internal
funzionamento: la funziona è interna e quindi non può essere richiamata da un utente, è accessibile solo da altre funzioni interne dello smart contract. La funzione Record è la funzione cardine dello smart contract in quanto è quella che effettivamente svolge la funzione di memorizzazione dei dati in blockchain.
In concreto la memorizzazione avviene tramite l'emissione di un evento "Recorded" che conterrà i dati identificativi del POD, i valori letti, il timestamp visto dal POD (che potrebbe essere differente dal momento della transazione in blockchain) e la tipologia di fonte del dato (inserimento manuale dell'utente, inserimento dal Signing POD, inserimento mediante messaggio firmato dal POD)

funzione: recordFromPod
input: uint _podId, bytes8 _sensorId, int32[] memory _sensorData, uint _when
output: -
visibilità: public
funzionamento: la funzione si fa carico di registrare i dati ricevuti da una transazione emessa direttamente dal POD tramite il suo indirizzo. La funzione controlla che l'indirizzo mittente sia quello corrispondente all' ID dichiarato per il POD, e che questo sia stato attivato da un utente proprietario.
Se queste condizioni sono verificate prende i dati passati e li registra in blockchain tramite la funzione record.

funzione: recordFromOwner
input: uint _podId, bytes8 _sensorId, int32[] memory _sensorData, uint _when
output: -
visibilità: public
funzionamento: la funzione si fa carico di registrare una i dati ricevuti da transazione realizzata manualmente dall'utente proprietario di un POD. La funzione controlla che il mittente sia un proprietario abilitato tramite la funzione enableAddress, e che abbia dichiarato la proprietà del POD tramite la funzione adopt.
Se queste condizioni sono verificate prende i dati passati e li registra in blockchain tramite la funzione record.

funzione: recordFromSigner
input: uint256 _nonce, address _podAddr, uint _podId, bytes8 _sensorId,
int32[] memory _sensorData, uint _when, bytes memory _signature
output: -
visibilità: public
funzionamento: la funzione si fa carico di registrare una i dati ricevuti da transazione emessa dall'utente proprietario di un POD che registra i dati firmati dal POD stesso. Il POD raccoglie i dati dai sensori, aggiunge il nonce, l'address, il proprio ID ed il timestamp. Con questi dati compone un messaggio, calcola l'hash del messaggio e con la propria chiave privata firma l'hash del messaggio. Tutti questi dati sono passati dal POD al suo proprietario. Il proprietario realizzerà la transazione da inviare allo smart contract.
La funzione controlla che il mittente sia un proprietario abilitato tramite la funzione enableAddress, e che abbia dichiarato la proprietà del POD tramite la funzione adopt. Passa quindi a valutare i dati ricevuti. Controlla che il nonce del pod sia valido (progressivo incrementale mai usato prima) e che la firma crittografica sia coerente con tutti gli altri dati e che quindi il messaggio sia corretto ed inalterabile.
Se queste condizioni sono verificate prende i dati passati e li registra in blockchain tramite la funzione record.

Eventi emessi (log)

Gli eventi sono log associati alle transazioni che li hanno fatti scaturire. Ogni operazione sugli smart contract avviene a partire da una chiamata fatta da un utente tramite una transazione firmata dal proprio wallet. La transazione, una volta minata e inserita in blockchain, può emettere eventi che restano scritti in blockchain nella "transaction receipt" in modo indelebile. Gli eventi sono utilizzati per lasciare una traccia evidente di quanto fatto dallo smart contract.

Nel nostro caso, gli eventi saranno lo strumento usato per depositare i dati di consumo dei pod e tutti i passaggi salienti del funzionamento dello smart contract e dell'uso che ne è stato fatto da amministratore e utenti.

E' possibile raccogliere i dati dai log sia in tempo reale che in un secondo momento. Apparentemente questo passaggio (mettere i dati sui log e poi successivamente prelevarli da li) potrebbe erroneamente sembrare un passaggio accessorio poco sensato. Il senso di questo passaggio è quello di lasciare traccia perenne ed inalterabile del funzionamento dello smart contract e dei valori letti così che nessuna diatriba o dubbio possa mai sollevarsi rispetto alla completezza e correttezza degli stessi.

Per raccogliere i dati sarà sufficiente utilizzare le API RPC messe a disposizione di un nodo per interrogarlo (vedi Lettura dei dati registrati)

evento: Enabled
parametri: address _who, address _wallet

significato: registra che l'utente identificato dai due address (_who, _wallet) è stato abilitato dalla funzione enableAddress

evento: Disabled
parametri: address _who
significato: registra che l'utente identificato dai due address (_who, _wallet) è stato disabilitato dalla funzione disableAddress

evento: Adopted
parametri: address _who, uint _pod, address _podaddress
significato: registra che l'utente con address _who ha dichiarato tramite una funzione adoptPod di possedere e gestire il POD con id _pod e con indirizzo _podaddress

evento: Rejected
parametri: address _who, uint _pod
significato: registra che l'utente con address _who ha dichiarato tramite una funzione reejctPod di disconoscere POD con id _pod precedentemente adottato

evento: Recorded
parametri: address _pod, uint _podId, bytes8 _sensorId, int32[] _data, uint _when, uint _mode
significato: è il messaggio fondamentale dello smart contract in quanto registra il dato più importante ovvero quello del consumo. I parametri dell'evento permettono di identificare lo specifico POD (tramite l'address _pod e l'identificativo _podId) lo specifico sensore del POD (_sensorId), la sequenza di dati (_data), il momento dichiarato dal POD (l'intero _when contiene il numero di secondi trascorsi dalle ore 0 e 0 minuti del 1° gennaio 1970). Si noti che il timestamp della notarizzazione può differire dal momento della transazione in cui essa è inserita.

Istanze di test

Di seguito i dati del deploy di una istanza dello smart contract sulla rete di test di Quadrans usati per i test e per la sperimentazione con le API.
Sarà possibile utilizzare i dati riportati per impostare un wallet e/o interagire con lo smart contract stesso.

Mittente e owner:

address: 0xaF1FEB6C52c98aF20b3ad4F3C3c9620C61De15E3
chiave privata :
0x886a0ea6664f68b17eb2b16f47c31265dffcb94122bdb5dbd543c405f0b350b1
link:
https://explorer.testnet.quadrans.io/address/0xaF1FEB6C52c98aF20b3ad4F3C3c9620C61De15E3/transactions

Contract

indirizzo del contract: 0xB7a6B6dA49893a00a5Ef768225cbf59f78E2Fd0E
https://explorer.testnet.quadrans.io/address/0xB7a6B6dA49893a00a5Ef768225cbf59f78E2Fd0E/transactions

transazione di deploy:
0xeca1c4129a04b16b18a0572bb6933b099fdcfaeb59b6c3fa1a2446fb78f21cc6
https://explorer.testnet.quadrans.io/tx/0xeca1c4129a04b16b18a0572bb6933b099fdcfaeb59b6c3fa1a2446fb78f21cc6/internal-transactions

Lettura dei dati

Una volta registrati i dati su blockchain tramite smart contract i dati si troveranno ad essere replicati su tutti i nodi della blockchain, siano essi direttamente afferenti al progetto o dedicati ad altri scopi da altri attori estranei al progetto. I dati saranno mantenuti a tempo indeterminato e non sarà necessario preoccuparsi del backup degli stessi.

Sarà possibile estrarre dalla blockchain tutti i dati registrati dagli smart contract utilizzando le primitive API RPC disponibili su un nodo e operabili tramite libreria **Web3.js** (https://web3js.readthedocs.io/en/v1.3.4/#)

Lettura manuale diretta dei dati

La procedura diretta dei dati è eseguibile manualmente appoggiandosi a diversi strumenti. Sebbene questa non sia la modalità consigliata e più comoda per eseguire la lettura, è opportuno conoscerla in quanto:

- viene eseguita dal singolo mediante strumenti pubblici non legati direttamente al progetto, e permette quindi un controllo terzo e separato rispetto a quello della piattaforma in caso di litigazioni
- può essere utilizzato come sistema di verifica e di debug in caso di problemi
- permette di realizzare sistemi di verifica indipendenti o strumenti di altra natura che possono lavorare sui dati inseriti in blockchain

La procedura di lettura manuale richiede di essere in possesso dell'indirizzo dello smart contract di notarizzazione. Questo solo dato, abbinato con le caratteristiche di tracciabilità e trasparenza della blockchain è sufficiente per ricavare tutti gli altri dati necessari.

Partendo dall'indirizzo dello smart contract è possibile vedere su un explorer (nel caso di quadrans abbiamo https://explorer.quadrans.io e https://explorer.testnet.quadrans.io) tutte le transazioni afferenti ad esso. E' quindi immediato passare all'interrogazione delle transazioni e leggere tutti i dati al loro interno. Dalle transazioni è infatti possibile ricavare:

- indirizzo del mittente delle transazioni
- operazioni svolte dall'indirizzo
- tipologia di mittente (SuperUser, utente abilitato, POD)

Sarà necessario disporre di un nodo della blockchain a cui rivolgere le interrogazioni tramite richieste RPC. In linea di massima una simile procedura userà le seguenti funzioni:

- **web3.eth.getBlock(blockHashOrBlockNumber)** permette di ricavare dati su un blocco dato il numero o l'hash del blocco. Tra queste le più rilevanti saranno il timestamp del blocco, il numero di transazioni contenute all'interno, e l'elenco delle stesse.
- le singole transazioni potranno essere recuperate in modi diversi con le funzioni **web3.eth.getTransactionFromBlock(hashStringOrNumber, indexNumber)** o **web3.eth.getTransaction(transactionHash)**
- una volta recuperata la singola transazione è possibile analizzarla rispetto ai suoi contenuti. Il valore del campo **to** dovrà coincidere con l'indirizzo dello smart contract

- in questo caso, mediante la **web3.eth.getTransactionReceipt(hash)** si potranno ricavare i dati relativi agli eventi/log in chiaro
- tutti i dati che è necessario leggere ed interpretare si trovano all'interno dell'array logs della transaction Receipt.

Lettura dei dati tramite API

La modalità di lettura consigliata è quella di utilizzare le API fornite con il progetto. Queste API permettono una facile integrazione con altri progetti in quanto sono un'interfaccia HTTP REST per la lettura dei dati più altre semplici utility.

Le API sono realizzate in linguaggio Javascript su piattaforma Node.JS e framework Express.

Le API fornite possono essere direttamente utilizzate in produzione, a condizione che siano prese tutte le contromisure di sicurezza a contorno da parte del sistemista che le mette in funzione. In alternativa possono essere utilizzato come riferimento per l'implementazione o l'estensione di una versione ad hoc o proprietaria.

Installazione ed avvio

Le API sono da installare su una macchina che già ospita un nodo della blockchain (Quadrans). Si consiglia di utilizzare una macchina linux su cui svolgere questi passaggi:

- scaricare il pacchetto del software delle API
- installare sulla macchina **node** e **npm** nella versione più recente
- installare le dipendenze delle API col comando **npm install**
- è possibile avviare e tenere monitorare le chiamate delle API in tramite i comandi 'npm start' o tramite lo strumento 'nodemon server.js'

Configurazione

Nella cartella config è presente il file config.js che contiene tutte le variabili ed i relativi valori per la messa in funzione delle API.

Uso delle API

Le API espongono tre rotte principali:

- **utils** : espone una serie di strumenti di comodo quali
    - creazioni di un wallet
    - lettura del saldo di un indirizzo
    - lettura dei dati di una transazione
    - creazione di un qrcode grafico

- **user** : permette registrare l'address ed il wallet di un utente per abilitarlo o di disabilitare un utente
- **pod** : esegue tutte le possibili operazioni sui dati di un pod. Questo permette di abilitare e disabilitare un pod, leggere dati relativi al pod come address, owner e nonce, e di effettuare registrazioni manuali o firmate
- **scan**: permette di cercare all'interno della blockchain transazioni relative allo smart contract per poi andare a leggerne il contenuto

Per una descrizione più precisa delle API si rimanda alla relativa documentazione in Appendice

Appendici

Appendice A: Testo Coordinato del Decreto-Legge 14 dicembre 2018, n. 135, articolo 8-ter

Tecnologie basate su registri distribuiti e smart contract

1. Si definiscono «tecnologie basate su registri distribuiti» le tecnologie e i protocolli informatici che usano un registro condiviso, distribuito, replicabile, accessibile simultaneamente, architetturalmente decentralizzato su basi crittografiche, tali da consentire la registrazione, la convalida, l'aggiornamento e l'archiviazione di dati sia in chiaro che ulteriormente protetti da crittografia verificabili da ciascun partecipante, non alterabili e non modificabili.
2. Si definisce «smart contract» un programma per elaboratore che opera su tecnologie basate su registri distribuiti e la cui esecuzione vincola automaticamente due o piu' parti sulla base di effetti predefiniti dalle stesse. Gli smart contract soddisfano il requisito della forma scritta previa identificazione informatica delle parti interessate, attraverso un processo avente i requisiti fissati dall'Agenzia per l'Italia digitale con linee guida da adottare entro novanta giorni dalla data di entrata in vigore della legge di conversione del presente decreto.
3. La memorizzazione di un documento informatico attraverso l'uso di tecnologie basate su registri distribuiti produce gli effetti giuridici della validazione temporale elettronica di cui all'articolo 41 del regolamento (UE) n. 910/2014 del Parlamento europeo e del Consiglio, del 23 luglio 2014.
4. Entro novanta giorni dalla data di entrata in vigore della legge di conversione del presente decreto, l'Agenzia per l'Italia digitale individua gli standard tecnici che le tecnologie basate su registri distribuiti debbono possedere ai fini della produzione degli effetti di cui al comma 3.

Riferimenti normativi

-Si riporta l'articolo 41 del regolamento (UE) n.9102014 del Parlamento europeo e del Consiglio, del 23 luglio 2014 in materia di identificazione elettronica e servizi fiduciari per le transazioni elettroniche nel mercato interno e che abroga la direttiva 1999/93/CE:

Art. 41. Effetti giuridici della validazione temporale elettronica

1. Alla validazione temporanea elettronica non possonoessere negati gli effetti giuridici e l'ammissibilita' come prova in procedimenti giudiziali per il solo motivo della sua forma elettronica o perche' non soddisfa i requisiti della validazione temporanea elettronica qualificata.

2. Una validazione temporale elettronica qualificata gode della presunzione di accuratezza della data e dell'ora che indica e di integrita' dei dati ai quali tale data e ora sono associate.

3. Una validazione temporale elettronica rilasciata in uno Stato membro e' riconosciuta quale validazione temporale elettronica qualificata in tutti gli Stati membri.

Link Gazzetta Ufficiale : https://www.gazzettaufficiale.it/eli/id/2019/02/12/19A00934/sg

Appendice B: Manuale delle API

Di seguito la documentazione delle API per la registrazione di consumi tramite pod e sensori su smart contract. Le API permettono di interagire con quasi tutte le funzioni dello smart contract tramite chiamate REST. Questo permette di facilitare lo sviluppo di interfacce esterne di diverso tipo disaccoppiando le diverse parti della soluzione. Tramite le API è possibile considerare la blockchain e gli smart contract un backend alla stregua di molti altri.

Questa stessa documentazione è disponibile online presso l'URL privato:
https://documenter.getpostman.com/view/5974971/Tzedgj3d

Esistono 4 rotte principali

- Utils contiene un set di utility che possono essere usati per facilitare diverse operazioni ed interrogazioni
- Pod contiene tutti gli endpoint per operare sui pod, aggiungerli o eliminarli, e leggerne i dati (stato, indirizzo, proprietario) e registrare i dati in due modi diversi
- User contiene gli endpoint per abilitare e disabilitare gli indirizzi degli utenti, proprietari dei pod
- Scan permette di cercare all'interno della blockchain le transazioni che fanno capo allo smart contract usato

Note sull'utilizzo Ogni qual volta si vede la dicitura {{base_url}} questa andrà sostituita con il percorso sotto cui le API sono state installate e possono rispondere Ex: https://localhost:8088

GET Utils/qr :

ex: {{base_url}}/utils/qr/data
Questa chiamata permette di generare un QRCode a partire da una stringa di dati. Il formato dei dati può influenzare il risultato. Il passato deve essere passato nel formato URL-Encoded.

Il campo ***data*** contiene i dati che saranno codificati nel QRCode

GET Utils/tx

ex: {{base_url}}/utils/tx/0xe948d5f03….9b4189cf376

Permette di leggere i dati di una transazione. Per leggere i dati è necessario passare la transaction hash come ultimo parametro (nell'esempio 0xe948d5f03...9b4189cf376). Il risultato è una struttura JSON con tutti i parametri in particolare:

- ***status*** è true quando la transazione è andata a buon fine
- ***timestamp*** il timestamp in formato UNIX del momento in cui la transazione è stata minata
- ***transaction*** i dati della transazione emessa in formato JSON
- ***receipt*** i dati della ricevuta della transazione eseguita in formato JSON.

Se la transazione non esiste, o non è stata ancora minata, i campi saranno tutti null

GET Utils/decode

ex: {{base_url}}/utils/decode/0xe948d5f03….9b4189cf376
Data una transazione, specificata dalla sua transaction hash, permette di recuperare i dati relativi agli eventi alla chiamata fatta. Si noti che la chiamata deve essere fatta verso il nostro smart contract o non saremo in grado di decodificarlo e riceveremo un errore.

In caso positivo la risposta conterrà un JSON con:

- la transaction hash
- il timestamp in formato UNIX
- i valori passati in input ai metodi chiamati
- i valori restituiti dagli eventi (log) emessi durante l'esecuzione

Nella nostra applicazione, ad esempio, può essere usata per ricavare i dati registrati in blockchain da un pod

GET Utils/balance

ex: {{base_url}}/utils/balance/0xaF1FEB6C52c98aF20b3ad4F3C3c9620C61De15E3
Dato l'address di un wallet, interroga la blockchain per capire il saldo. Questa funzione è utile per monitorare gli wallet usati, controllare se è necessario integrare la loro disponibilità per effettuare operazioni future etc.

GET Utils/wallet

ex: {{base_url}}/utils/wallet/asda

Crea o ottiene un nuovo wallet in due modi diversi

in modo deterministico : passando una segreto come ultimo parametro, inquesto caso asda, le API derivano dalla stringa una chiave privata ed un address.
in modo totalemte randomico: in questo caso le API genereranno un numero casuale e da quello ricaveranno chiave privata e address. Ogni chiamata avrà valori diversi non ripetibili
La procedura ritorna un JSON con:

- chiave privata
- address
- segreto usato (se inviato nella chiamata)

POST Pod/record

ex: {{base_url}}/pod/record
Questa è forse la funzione più importante, quella che permette di registrare i consumi letti da un POD.

Lo smart contract prevede 3 modalità di scrittura:

- diretta dal pod
- manuale da parte del proprietario sotto sua responsabilità
- veicolata dal proprietario tramite dati firmati dal pod

Questa stessa chiamata permette di effettuare le ultime 2 a seconda dei parametri che sono passati

Caso scrittura manuale fatta con dati firmati

In questo caso il proprietario dovrà inviare

- ***nonce*** : il numero progressivo passato dal pod per le sue scritture, deve essere strettamente crescente
- ***signature***: firma digitale composta da un json di tre campi r,s,v
- ***podAddress*** : indirizzo del pod (esadecimale, 20byte)
- ***podId*** : valore numerico a 256 bid
- ***sensorId***: 8 byte in esadecimale
- ***sensorData*** un array di dati letti dal sensore come interi a 32 bit
- il ***timestamp*** della lettura (anteriore a quella di registrazione)
- ***priv***: la propria chiave privata con cui firmare la transazione
- il parametro ***mined*** (true/false) per indicare se si vuole attendere il mining della transazione o solo la trasnaction hash

Caso scrittura manuale fatta dal proprietario

In questo caso il proprietario dovrà inviare

- ***podId*** : valore numerico a 256 bid

- ***sensorId***: 8 byte in esadecimale
- ***sensorData****:* un array di dati letti dal sensore come interi a 32 bit
- il ***timestamp*** della lettura (anteriore a quella di registrazione)
- **priv**: la propria chiave privata con cui firmare la transazione
- il parametro ***mined*** (true/false) per indicare se si vuole attendere il mining della transazione o solo la trasnaction hash

Risposta

A seconda del valore impostato per il parametro ***mined***, si possono avere 2 risultati diversi

- se ***mined*** è true la procedura attenderà che la transazione sia minata in blockchain (potrebbe richiedere diverso tempo e far andare in timeout la sessione http) e ritornerà tutti i dati della transazione
- se ***mined*** è false la procedura ritornerà immediatamente la transaction hash. Sarà poi possibile usare quella transaction hash per chiedere in modo asincrono l'esito della tranaszione tramite la chiamata utils/tx e utils/decode
- e ricevere tutti i dati (nel caso true), o se si preferisce non attendere e ricevere solo la transaction hash da controllare successivamente (nel caso false)

Esempio di chiamata coi parametri Body urlencoded

nonce	2
podAddress	0xE8e5FF5C6a50ab903972f16Bb60F974d3a4A7f7b
podId	1
sensorId	0x0e8b2788104c5856
sensorData	15137
timestamp	1624463938
signature	{"r": "0x859b26748166f4287a743d4cb34338e24135a0e267c753a7021766340b89cf3e", "s": "0x39b8d77c05b5a4ecd24afc253136600e09de73fe35a7fbe7f8723ad00b9da1b2", "v": 28}
priv	886a0ea6664f68b17eb2b16f47c31265dffcb94122bdb5dbd543c405f0b350b1
mined	true

POST Pod/adopt

ex: {{base_url}}/pod/adopt
Permette ad un utente abilitato di dichiarare il possesso di uno specifico pod e di poter poi disporre dei suoi dati. La chiamata deve essere fatta dall'utente stesso.

Vengono passati i parametri

- ***podId*** : valore numerico a 256 bid
- ***podAddress***: l'indirizzo del pod (esadecimale 20 byte)
- **priv**: la propria chiave privata con cui firmare la transazione
- il parametro ***mined*** (true/false) per indicare se si vuole attendere il mining della transazione o solo la trasnaction hash

Risposta
A seconda del valore impostato per il parametro ***mined***, si possono avere 2 risultati diversi

- se ***mined*** è true la procedura attenderà che la transazione sia minata in blockchain (potrebbe richiedere diverso tempo e far andare in timeout la sessione http) e ritornerà tutti i dati della transazione
- se ***mined*** è false la procedura ritornerà immediatamente la transaction hash. Sarà poi possibile usare quella transaction hash per chiedere in modo asincrono l'esito della tranaszione tramite la chiamata utils/tx e utils/decode

Esempio di chiamata coi parametri Body urlencoded

podAddress	0xE8e5FF5C6a50ab903972f16Bb60F974d3a4A7f7b
podId	1
priv	886a0ea6664f68b17eb2b16f47c31265dffcb94122bdb5dbd543c405f0b350b1
mined	true

POST Pod/reject

ex: {{base_url}}/pod/reject
Permette ad un utente abilitato di disconoscere un pod precedentemnete adottato. La chiamata deve essere fatta dall'utente stesso.

Vengono passati i parametri

- ***podId*** : valore numerico a 256 bid
- ***priv:*** la propria chiave privata con cui firmare la transazione

- il parametro ***mined*** (true/false) per indicare se si vuole attendere il mining della transazione o solo la trasnaction hash

Risposta

A seconda del valore impostato per il parametro ***mined***, si possono avere 2 risultati diversi

- se ***mined*** è true la procedura attenderà che la transazione sia minata in blockchain (potrebbe richiedere diverso tempo e far andare in timeout la sessione http) e ritornerà tutti i dati della transazione
- se ***mined*** è false la procedura ritornerà immediatamente la transaction hash. Sarà poi possibile usare quella transaction hash per chiedere in modo asincrono l'esito della tranaszione tramite la chiamata utils/tx e utils/decode

Esempio di chiamata coi parametri Body urlencoded

podId	1
priv	886a0ea6664f68b17eb2b16f47c31265dffcb94122bdb5dbd543c405f0b350b1
mined	true

GET Pod/nonce

ex: {{base_url}}/pod/nonce/1
Permette di recuperare dallo smart contract il valore attuale del nonce di un pod, ovvero il contatore numerico crescente delle registrazioni firmate da quel pod. Si noti che il nonce cresce solo in concomitanza di una registrazione fatta mediante dati firmati. L'ultimo parametro della chiamata è l'identificativo numerico del pod, in queso caso 1

GET Pod/address

ex: {{base_url}}/pod/address/1
Permette di recuperare dallo smart contract il valore dell'address di un pod come da precedente chiamata adpopt. L'ultimo parametro della chiamata è l'identificativo numerico del pod, in queso caso 1

GET Pod/owner

ex: {{base_url}}/pod/owner/1
Permette di recuperare dallo smart contract il proprietario di un pod, ovvero colui che ha precedente chiamato adopt con quell' identificativo numerico per il pod. L'ultimo parametro della chiamata è l'identificativo numerico del pod, in questo caso 1

POST User/enable

ex: {{base_url}}/user/enable/
Questa chiamata può essere fatta solo dal gestore della piattaforma in quanto la transazione verrà firmata col suo address.
La procedura permette di abilitare un utente all'uso della piattaforma. L'utente è identificato tramite 2 address, il primo ***address*** sarà quello usato per firmare le transazioni, il secondo ***wallet*** viene lasciato per futuri utilizzi come spiegato nella documentazione

il parametro ***mined*** (true/false) per indicare se si vuole attendere il mining della transazione o solo la trasnaction hash

Risposta

A seconda del valore impostato per il parametro ***mined***, si possono avere 2 risultati diversi

- se ***mined*** è true la procedura attenderà che la transazione sia minata in blockchain (potrebbe richiedere diverso tempo e far andare in timeout la sessione http) e ritornerà tutti i dati della transazione
- se ***mined*** è false la procedura ritornerà immediatamente la transaction hash. Sarà poi possibile usare quella transaction hash per chiedere in modo asincrono l'esito della tranaszione tramite la chiamata utils/tx e utils/decode

Esempio di chiamata coi parametri Body urlencoded

address	0xf8e869bee6579d86a6d750c3ddb93fb5111bde6a
wallet	0x421a652f91e8b84f78dc7a0a159dbdc024cfdc0a
mined	true

POST User/disable

ex: {{base_url}}/user/disable/

Questa chiamata può essere fatta solo dal gestore della piattaforma in quanto la transazione verrà firmata col suo address. La procedura permette di disabilitare un utente all'uso della piattaforma. L'utete è identificato tramite ***l'address*** dichiarato nella funzione ***enable***

il parametro ***mined*** (true/false) per indicare se si vuole attendere il mining della transazione o solo la trasnaction hash

Risposta

A seconda del valore impostato per il parametro mined, si possono avere 2 risultati diversi

- se ***mined*** è true la procedura attenderà che la transazione sia minata in blockchain (potrebbe richiedere diverso tempo e far andare in timeout la sessione http) e ritornerà tutti i dati della transazione
- se ***mined*** è false la procedura ritornerà immediatamente la transaction hash. Sarà poi possibile usare quella transaction hash per chiedere in modo asincrono l'esito della tranaszione tramite la chiamata utils/tx e utils/decode

Esempio di chiamata coi parametri Body urlencoded

address	0xf8e869bee6579d86a6d750c3ddb93fb5111bde6a
mined	true

GET Scan

ex: {{base_url}}/scan/4550000/4557000
Questa chiamata permette cercare all'interno della blockchain tutte le chiamate fatte allo smart contract dell'applicazione nelle rispettive transazioni.

Richiede di specificare

- un primo parametro from (in questo caso 4550000) indicante il numero del blocco da cui cercare
- un parametro to opzionale (in questo caso 4557000) indicante il numero del blocco fino a cui cercare. Se non specificato si applica automaticamente il limite massimo di blocchi possibili come definito nel file di configurazione.

La procedura fa la scansione di tutti i blocchi nell'intervallo di riferimento, cerca transazioni destinate allo smart contract (NB: le letture non comportano transazioni) e riporta la transaction hash corrispondente

Risposta

Viene restituito un oggetto JSON con:

- indicazione del blocco di partenza usato
- indicazione del blocco di fine effettivo
- un array di elemnti trovati in cui ogni elemento sarà a sua volta un oggetto JSON con:
    - il numero del blocco
    - il timestamp del blocco
    - l'hash della transazione
    - il mittente della transazione

Appendice C: Elenco software consegnato

Oltre alla presente documentazione viene consegnato il seguente sofware suddiviso per cartelle:

```
├── Energy Meters.pdf                          | Questa
documentazione
└── Software                                   |
    ├── API                                    | API REST
smartcontract
    │   ├── auth                               |
    │   ├── common                             |     Librerie API
    │   │   ├── bctools.js                     |
    │   │   ├── database.js                    |
    │   │   ├── logger.js                      |
    │   │   └── tools.js                       |
    │   ├── config                             |    File di
    │   │   ├── config.js                      |    Configurazione
    │   │   └── contract.abi                   |    ABI Smart
Contract
    │   ├── ENERGY.postman_collection.json     | Chiamate API
    │   ├── logs                               | Cartela dei log
    │   ├── package.json                       |
    │   ├── package-lock.json                  |
    │   ├── routes                             | Rotte Statiche API
    │   │   ├── podroute.js                    |
    │   │   ├── scanroute.js                   |
    │   │   ├── userroute.js                   |
    │   │   └── utilsroute.js                  |
    │   ├── server.js                          | File principale API
    │   └── SSL                                | Certificati SSL
    |                                          |
    ├── SmartContracts                         |
    │   ├── consumption.sol                    | Smart Contract
    │   ├── deploy.txt                         | Installazione di
test
    │   ├── Migrations.sol                     | Script di deploy
    │   └── wallet_data.txt                    | Wallet di test
    └── Unit_test                              | Test (truffle)
        └── 01_test.js                         |
```

Autorizzazione

Autorizza, l'Università degli Studi di Perugia e i proponenti del progetto SIN_00968, senza limiti di tempo, anche ai sensi degli artt. 10 e 320 cod.civ. e degli artt. 96 e 97 legge 22.4.1941, n. 633, Legge sul diritto d’autore, alla pubblicazione, modifica e/o diffusione in qualsiasi forma del presente elaborato e prende atto che la finalità di tali pubblicazioni sono meramente di carattere informativo ed eventualmente promozionale.