



Installazione, configurazione e amministrazione.

Manuale d'uso

versione 1.0 Beta

Luglio 2020.

MiniBlocklyChain è un marchio registrato di OpenQbit Inc., sotto licenza d'uso gratuito e commerciale. Termini e condizioni d'uso all'indirizzo: www.OpenQbit.com

Contenuto

1.	Introduzione.....	3
2.	Che cos'è una rete pubblica o privata nello schema a catena?	4
3.	Cos'è la programmazione Blockly?.....	4
4.	Che cos'è Termux?	5
5.	Cos'è Mini BlocklyChain?.....	5
6.	Architettura di processo in Mini BlocklyChain	9
7.	Diagramma di funzionalità BlocklyChain (Mini BlocklyChain).....	12
8.	Cos'è il Mini BlocklyCode?.....	13
9.	Installazione della rete di comunicazione Mini BlocklyChain	14
10.	Sincronizzazione nei nodi del sistema (telefono cellulare) minuti e secondi.	33
11.	Configurazione della memoria all'interno di Termux.	41
12.	Installazione di rete "Tor" e installazione "Syncthing".....	42
13.	Installazione del database "Redis" e del server SSH (Secure Shell).	43
14.	Configurazione del server SSH su cellulare (smartphone).	44
15.	Configurazione di rete "Tor" con servizio SSH (Secure Shell).	50
16.	Configurazione del sistema Peer to Peer con sincronizzazione manuale.	53
17.	Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).	64
18.	Che cos'è la Prova del Quantum (PQu)?	65
19.	Definizione e uso dei blocchi in Mini BlocklyChain	68
20.	Utilizzo di blocchi per database SQLite (versione MiniSQLite)	94
21.	Definizione e uso dei blocchi di sicurezza.	98
22.	Impostazione dei parametri di sicurezza in Mini BlocklyChain.....	108
23.	Allegato "Creazione di banche dati KeyStore & PublicKeys".	112
24.	Allegato "Comandi RESTful SQLite GET/POST".	125
25.	Allegato "Connettore Java Code SQLite-Redis Connector".....	130
26.	Allegato "Mini BlocklyChain per sviluppatori".	134
27.	Allegato "Contratti intelligenti BlocklyCode".....	146
28.	Allegato "OpenQbit Quantum Computing".	152
29.	Allegato "Blocchi estesi per database SQLite".....	156
30.	Allegato "Esempio di creazione di un sistema Mini BlocklyChain".	156
31.	Allegato "Integrazione con gli ambienti Ethereum & Bitcoin".....	180
32.	Licenze e utilizzo del software.	199

1. Introduzione.

La blockchain è generalmente associata a Bitcoin e ad altre valute crittografiche, ma queste sono solo la punta dell'iceberg, poiché non viene utilizzata solo per la moneta digitale, ma può essere utilizzata per qualsiasi informazione che possa avere un valore per gli utenti e/o le aziende. Questa tecnologia, che ha le sue origini nel 1991, quando Stuart Haber e W. Scott Stornetta hanno descritto il primo lavoro su una catena di blocchi criptografati, è stata notata solo nel 2008, quando è diventata popolare con l'arrivo del bitcoin. Ma attualmente il suo utilizzo è richiesto in altre applicazioni commerciali e si prevede che crescerà nel medio futuro in diversi mercati, come le istituzioni finanziarie o l'Internet degli oggetti tra gli altri settori.

La blockchain, meglio conosciuta con il termine blockchain, è un singolo record concordato distribuito su più nodi (dispositivi elettronici come PC, smartphones, tablet, ecc.) in una rete. Nel caso delle valute criptate, possiamo pensarlo come il libro contabile in cui viene registrata ciascuna delle operazioni.

Il suo funzionamento può essere complesso da capire se entriamo nei dettagli interni della sua realizzazione, ma l'idea di base è semplice da seguire.

Viene memorizzato in ogni blocco:

1.- una serie di registrazioni o transazioni valide,

2.- informazioni relative a quel blocco,

3.- il suo collegamento con il blocco precedente e il blocco successivo attraverso l'hash di ogni blocco –un codice unico che sarebbe come l'impronta digitale del blocco.

Pertanto, **ogni blocco ha un posto specifico e non mobile all'interno della catena**, in quanto ogni blocco contiene informazioni provenienti dall'hash del blocco precedente. L'intera catena è memorizzata su ogni nodo di rete che compone la blockchain, quindi **una copia esatta della catena è memorizzata su tutti i partecipanti alla rete**.

Man mano che vengono creati nuovi record, questi vengono prima controllati e convalidati dai nodi della rete e poi aggiunti ad un nuovo blocco che viene collegato alla catena.

Ora immaginate che questa rete di dispositivi che comunicano tra loro, ha la capacità di interagire senza l'intervento di una persona, cioè il grande vantaggio della blockchain è che può prendere decisioni autonome, che beneficia in tempi di risposta del servizio agli utenti, disponibile 24 ore su 24, minimizza i costi nel business e prima di tutto ha un livello di sicurezza già testato, per questo e per altri motivi è diventato così popolare in uso in diversi settori pubblici e privati.

2. Che cos'è una rete pubblica o privata nello schema a catena?

Rete pubblica. - Si tratta di una rete di computer o dispositivi mobili che comunicano tra loro e mantengono l'anonimato, non si sa chi interagisce in questa rete in modo formale nelle loro transazioni, in questo tipo di rete qualsiasi persona o azienda può interagire e connettersi in qualsiasi momento perché non sono necessari i permessi per connettersi, un esempio è la blockchain di Bitcoin, chiunque può entrare per comprare o vendere. Normalmente questo tipo di reti sono orientate o dirette all'acquisto e alla vendita di beni monetari digitali o al suo sinonimo di "crittomoni", esempi: DogCoin, Ethereum, LiteCoin, BitCoin, Onde, ecc.

Rete privata. - Si tratta di una rete di computer o dispositivi mobili che comunicano tra loro. Tuttavia, a differenza delle reti pubbliche, le reti private necessitano di una previa autorizzazione da parte di qualche ente (azienda o persona) per potersi collegare ed essere parte di questo tipo di rete. Normalmente, le reti private a catena di blocco sono utilizzate in aziende o società per effettuare transazioni o operazioni con una varietà di diversi tipi di informazioni che possono avere un valore tangibile sotto forma di documenti, processi, autorizzazioni e/o decisioni commerciali applicate e supervisionate da catene di blocco, esempi: settore finanziario, settore assicurativo, governo, tra gli altri.

3. Cos'è la programmazione Blockly?

Blockly è un **linguaggio di programmazione visuale** composto da un semplice insieme di comandi che possiamo combinare come se fossero i pezzi di un puzzle. È uno strumento molto utile per chi vuole **imparare a programmare** in modo intuitivo e semplice o per chi sa già programmare e vuole vedere le potenzialità di questo tipo di programmazione.

Blockly è una forma di programmazione in cui non è necessario alcun background in nessun tipo di linguaggio informatico, questo perché si tratta solo di unire blocchi grafici come se stessimo giocando a lego o a un puzzle, basta avere un po' di logica ed è tutto!

Chiunque può creare programmi per cellulari (smartphone) senza pasticciare con quei linguaggi di programmazione difficili da capire, basta mettere insieme i blocchi in modo grafico in modo semplice, facile e veloce da creare.

4. Che cos'è Termux?

Termux è un emulatore di terminale Android e un'applicazione per l'ambiente Linux che funziona direttamente senza bisogno di routing o configurazione. Un sistema di base minimo viene installato automaticamente.

Useremo Termux per la sua stabilità e la sua facile installazione e gestione, tuttavia, è possibile utilizzare un ambiente installato di Ubuntu Linux per Android.

In questo ambiente Linux avrete il "cuore" dei processi di comunicazione del MiniBlocklyChain.

5. Cos'è Mini BlocklyChain?

Mini BlocklyChain è una blockchain completamente funzionale è una tecnologia sviluppata per i telefoni cellulari con sistema operativo (OS) **Android** che saranno i nodi che si esibiranno nell'invio e nella ricezione delle transazioni. Abbiamo creato la prima tecnologia blockchain che è strutturata in modo "modulare" attraverso la programmazione Blockly dove chiunque abbia una conoscenza minima e senza saper programmare può creare e sviluppare programmi per cellulari e creare la propria blockchain sia in modalità rete pubblica che privata. Se si vuole creare la propria moneta digitale è possibile farlo o una soluzione per utilizzarla in un'azienda, le possibilità si basano sulle esigenze di ogni caso reale e sulla logica di business che l'utente adotta o crea in base alle proprie esigenze.

Mini BlocklyChain è il primo blockchain modulare per telefoni cellulari in cui un sistema transazionale può essere implementato in breve tempo.

Prima di entrare nella definizione e nell'uso dei blocchi "modulo" dobbiamo avere i concetti di base dei componenti Mini BlocklyChain per sapere quando, come e dove applicarli secondo il caso reale che vogliamo implementare.

I seguenti concetti sottolineano il tipo di utente a cui si rivolgono, quindi non è importante per le persone che non hanno competenze di programmazione che i concetti per gli utenti dello sviluppo siano pienamente compresi.

Concetti di base:

Nodo. - Ogni dispositivo mobile (telefono, tablet, ecc.) con sistema operativo Android che fa parte della rete pubblica o privata di Mini BlocklyChain viene nominato come nodo di questa rete.

Hashish. - Si tratta di una firma digitale che viene associata a un insieme di dati, stringa di caratteri, documenti o qualche tipo di informazione digitale, questa firma digitale è unica e

irripetibile che aiuta a inviare o ricevere informazioni in modo che il contenuto iniziale delle informazioni inviate non possa essere alterato.

Attivo. - Qualsiasi tipo di dato o informazione digitale che può essere ponderato con un certo valore materiale o immateriale per persone o aziende (documenti, monete digitali, musica, video, immagini, autorizzazioni digitali, ecc.)

Transazione. - Scambio di informazioni tra nodi che occupano un qualche tipo di asset.

UXTO Transaction - È un tipo di transazione che impacchetta una o più transazioni dai nodi e tutte le transazioni non spese da ogni nodo (UXTO: Unspent *Transaction Outputs*) possono essere spese come input in una nuova transazione. Queste transazioni sono raggruppate in una coda da elaborare ogni volta che si può regolare secondo i requisiti di ogni flusso di informazioni.

Transazione (ingresso). - È un tipo di transazione che si basa sulle transazioni UXTO. Quando una coda di transazioni UXTO entra, viene elaborata da una transazione di **input** che classifica la transazione come un deposito o una spesa e quindi è in grado di avere un saldo del risultato finale per ogni utente.

Indirizzo della fonte. - Questo è l'indirizzo della persona che genera o invia la transazione da elaborare nella coda SQLite Master. Si tratta di un numero alfanumerico di 64 caratteri che viene creato e verificato dal sistema.

Indirizzo di destinazione. -Questo è l'indirizzo della persona che riceve la transazione e la aggiunge al suo saldo o memorizza il bene inviato. Si tratta di un numero alfanumerico di 64 caratteri che viene creato e verificato dal sistema.

SQLite Master. - Database API REST che riceve le transazioni e crea una coda da elaborare ogni volta che una determinata variabile o un determinato tempo fisso a seconda delle esigenze aziendali.

Transazione DataBase. - Sono le transazioni che si riflettono nel database SQLite che riserva o memorizza le informazioni sulle transazioni Mini BlocklyChain.

AES (Advanced Encryption Standard) - Si tratta di un processo di sicurezza che critta i dati memorizzati nel database SQLite di Mini BlocklyChain.

Equilibrio. - Dopo aver effettuato qualsiasi processo di transazione nel Mini BlocklyChain, si ottiene un bilancio dell'operazione sia come valore tangibile (criptomoneta) che come valore intangibile (processi di business tra persone o aziende).

Processo di business. - Si tratta di qualsiasi processo che comporta un qualche tipo di flusso di informazioni per ottenere un risultato per un utente finale, l'utente finale può essere una persona o un'azienda di una varietà di settori nel settore privato o pubblico.

Chiave pubblica e privata. - Si tratta di un tipo di crittografia delle informazioni in cui vengono generate due chiavi alfanumeriche associate tra loro e utilizzate per inviare informazioni

sensibili attraverso reti pubbliche o private. La chiave privata viene utilizzata per criptare le informazioni e quando viene inviata attraverso la rete non può essere alterata o essere leggibile a prima vista, la chiave pubblica è quella che viene condivisa e viene vista da qualsiasi persona o azienda e questa aiuterà a verificare le informazioni inviate, oltre che a poterle leggere solo dal destinatario valido.

Firma digitale. - Si tratta di una firma che può essere creata con la chiave privata, con questa firma il destinatario assicura che le informazioni non sono state alterate e conferma che le informazioni sono valide per il destinatario.

Indirizzo digitale (indirizzo Mini BlocklyChain). - Si tratta di un indirizzo che è composto da caratteri alfanumerici unici per ogni utente di Mini BlocklyChain, questo indirizzo viene utilizzato per effettuare transazioni tra utenti e rete pubblica o privata.

Numero magico. - Questo è un numero casuale definito dalle regole di business per ogni transazione UXTO in esecuzione che serve ad autorizzare il nodo ad essere un candidato ad eseguire la transazione UXTO e a creare un nuovo blocco il Mini BlocklyChain.

Creazione di un nuovo blocco. - Un nuovo blocco in Mini BlocklyChain è un hash creato e attaccato dal nodo che è uscito come candidato vincente per elaborare la transazione UXTO.

Protocollo di consenso PoW (Proof of Work). - E' un test che esegue tutti i nodi e il primo nodo che termina il test con successo è quello scelto per eseguire la transazione UXTO. Si tratta di un algoritmo che si compone di calcoli matematici per ottenere un risultato (hash) secondo regole date in ogni transazione eseguita da Mini BlocklyChain si basa sull'usura o sulla richiesta delle risorse informatiche dei computer, nel caso di Mini BlocklyChain è stato strutturato e modificato per ottenere un "numero magico" questo è un numero per poter avere l'autorizzazione o il consenso della maggioranza dei nodi per essere quello che può eseguire la transazione UXTO. Il PoW ha un livello di difficoltà massimo di 5 poiché viene utilizzato solo per ottenere il "numero magico".

Protocollo di consenso PoQu (test quantistico) - Si tratta di un test che esegue tutti i nodi e che è composto inizialmente dal PoW per ottenere il "numero magico" e successivamente esegue un algoritmo di probabilità basato su un QRNG (Quantum Random Number Generator) un generatore di numeri casuali quantistici, questo processo assicura l'uguaglianza di probabilità per tutti i nodi e quindi sceglie quale nodo sarà quello che eseguirà la transazione UXTO e la creazione del nuovo blocco.

L'albero di Merkle. - Questo è un metodo di sicurezza per assicurare a Mini BlocklyChain che le transazioni sono valide o non sono state modificate da alcun ente esterno. Un hash merkle tree è una struttura ad albero di dati binari o non binari in cui ogni nodo che non è un foglio è etichettato con l'hash della concatenazione delle etichette o dei valori dei suoi nodi figli. Sono una generalizzazione delle liste di hash e delle stringhe di hash.

Redis DB. - Database delle transazioni in tempo reale utilizzato per elaborare e inviare ai nodi le nuove transazioni richieste.

SQLite DB. - Database in cui sono archiviati i bilanci e i nuovi blocchi per Mini BlocklyChain per garantire l'integrità della rete.

Sentinella. - Connettore di sicurezza e integrità dei dati tra Redis e SQLite. Questo connettore o programma è incaricato di esaminare, elaborare, convalidare, distribuire e tradurre le transazioni ai nodi.

OpenSSH. - Connettore di sicurezza per eseguire compiti all'interno del sistema operativo Android.

Terminale a conchiglia Termux. - Programma dove si possono trovare dipendenze di terze parti per elaborare le transazioni di Mini BlocklyChain, in questa versione 1.0 viene utilizzato per una facile installazione ma nelle versioni future sarà sostituito dal sistema BlocklyShell che è attualmente in fase di sviluppo.

Portafoglio. - È il repository digitale dove sono conservati due dati fondamentali in ogni transazione; le chiavi pubbliche e private che saranno utilizzate rispettivamente come indirizzo di origine e firma digitale per ogni transazione eseguita, nonché il saldo delle transazioni inviate o ricevute. Si tratta di un repository dove sono conservati gli indirizzi Mini BlocklyChain e i risultati delle transazioni UXTO (Balance).

Sviluppatore o programmatore di applicazioni:

Programmazione orientata agli oggetti (Java) - Mini BlocklyChain è realizzato in Java.

BlocklyCode. - È il termine dei contratti intelligenti che possono essere eseguiti nell'ambiente Mini BlocklyChain, il BlocklyCode viene creato nella programmazione java.

OpenJDK per Android. - È la suite di JDK e JRE per Android dove vengono creati ed eseguiti i BlocklyCode.

QRNG (Generatore di numeri casuali quantistici). - Si tratta di un generatore di numeri casuali quantistici, Mini BlocklyChain ha attualmente due API per la generazione di questi.

rsync. - Si tratta di un'applicazione gratuita per sistemi di tipo Unix e Microsoft Windows che offre un'efficiente trasmissione di dati incrementali, che funziona anche con dati compressi e criptati.

ffsend. - È un'applicazione per condividere i file in modo semplice e sicuro dalla linea di comando.

NTP. - Network Time Protocol, è un protocollo Internet per la sincronizzazione degli orologi dei sistemi informatici attraverso il routing dei pacchetti in reti a latenza variabile.

Termux (sviluppo). - Terminale a guscio con una vasta gamma di applicazioni opensource e librerie.

Moduli a blocchi (dati). - Gli sviluppatori possono integrare moduli per migliorare le caratteristiche di Mini BlocklyChain.

Da pari a pari. - Con questo termine si intende la comunicazione tra i nodi in modo diretto, cioè l'aggiornamento delle informazioni non dipende da un server centrale, ma ogni nodo funziona come un server centrale che comunica tra tutti i nodi che hanno le stesse informazioni, il che aiuta ad evitare punti di guasto.

Sincronizzazione. - strumento per la sincronizzazione di dati o file tra due dispositivi utilizzando il tipo di comunicazione "Peer to Peer".

Red Tor. - Si tratta di una rete di comunicazione distribuita a bassa latenza sovrapposta su Internet, in cui l'instradamento dei messaggi scambiati tra gli utenti non rivela la loro identità, ossia il loro indirizzo IP (anonimato a livello di rete).

Routing mobile. - Processo di installazione di software esterno sul telefono per entrare come amministratore di sistema nei sistemi operativi Linux (Android) l'utente amministratore è chiamato "root" per poter ruotare il telefono avrà accesso a qualsiasi processo. È importante notare che alcuni produttori di telefoni cellulari (Smartphone) affermano che la garanzia viene persa a causa di un eventuale difetto del telefono cellulare.

6. Architettura di processo in Mini BlocklyChain

Mini BlocklyChain è formato da tre processi che formano la sua architettura, il primo è il processo di business, il secondo è il processo di comunicazione e il terzo è l'ambiente di sviluppo di moduli complementari e/o la creazione di "contratti intelligenti" BlockyCode.

I processi di business sono i blocchi che formano una serie di routine per creare una transazione sia in rete pubblica che privata, questo tipo di processo è la pianificazione del business, il come, quando, cosa, chi, dove e più attributi saranno ordinati, pianificati e distribuiti in modo da ottenere la funzionalità principale di ogni transazione inviata, ricevuta, memorizzata e/o rifiutata. Il primo processo è composto dai seguenti tipi di blocchi suddivisi in quattro categorie:

1. Blocchi di inserimento dati
2. Blocchi di elaborazione dati
3. Blocchi di sicurezza.
4. Blocchi di dati modulari (sviluppatori)
5. Bot di comunicazione.

I blocchi di immissione dati sono quelli che ricevono la transazione con un minimo di quattro parametri di immissione (**indirizzo sorgente, indirizzo di destinazione, attivo, dati**) e possono averne di più a seconda della variabile "dati", questo può essere un blocco sviluppato da terzi o con un valore nullo (NULL).

I blocchi di processo dei dati sviluppano la logistica, il calcolo, la normalizzazione dei dati, le decisioni logiche e i controlli di flusso per ogni transazione. Questi tipi di blocchi sono utilizzati per dare intelligenza al processo di business e si basano principalmente come indica il suo nome sull'elaborazione di tutti i tipi di informazioni, così come la loro conversione da diversi tipi di dati.

I blocchi di sicurezza sono utilizzati per convalidare le informazioni e garantire che le transazioni non siano state alterate dalla loro origine alla loro destinazione, sono sempre elaborati con vari algoritmi di sicurezza utilizzati nelle tecnologie a catena di blocchi, tra gli strumenti più utilizzati sono (firma hash, firma digitale, crittografia dei dati AES, creazione e validazione di indirizzi digitali, tra gli altri).

I blocchi dati modulari, questi sono i blocchi che vengono sviluppati da terze parti, ricordiamo che Mini BlocklyChain è stato creato in modo modulare per essere arricchito da nuovi blocchi secondo le esigenze di ogni settore sia pubblico che privato. Per saperne di più su come creare i moduli, consultare l'allegato per gli sviluppatori.

Come abbiamo commentato in precedenza l'architettura di Mini BlocklyChain nella sua seconda componente sono i processi di comunicazione, questi processi sono quelli che si occupano dei canali di comunicazione via TCP e Sockets di comunicazione per dare flessibilità di invio e ricezione delle transazioni sia con i diversi mezzi che vengono utilizzati al momento i più utilizzati: Internet mobile, Wifi attualmente al lavoro per includere il mezzo di comunicazione Bluetooth.

I blocchi di comunicazione si basano fondamentalmente sullo scambio di dati con la sicurezza implementata nel canale di trasferimento e questo si basa su una comunicazione attraverso il protocollo SSH (Secure Shell) con le diverse fasi che una transazione inviata o ricevuta attraversa.

La parte di comunicazione è fondamentale in quanto questi processi hanno la funzione di aggiornare le informazioni in tutti i nodi via TCP e la connessione chiamata "**Peer to Peer**" i blocchi che intervengono nella comunicazione si basano sullo scambio di informazioni tra i nodi senza l'intervento di server intermediari in modo che ogni nodo renda indipendente il poter creare una rete di nodi dove i punti di guasto sono minimi o quasi nulli. Allo stesso modo, questa indipendenza di ogni nodo li aiuta a prendere decisioni individualmente o nel loro insieme secondo le esigenze dell'azienda.

L'architettura "Peer to Peer" è composta da tre parti che formano la rete pubblica o privata che si decide di creare, in entrambi i casi il canale di comunicazione è criptato da nodo a nodo.

Il primo componente per la comunicazione tra dispositivi mobili (smartphone) o wifi è quello di fornire ai nodi o dispositivi una rete in cui si possono trovare in qualsiasi parte del mondo, la rete di comunicazione in cui è montato MiniBlocklyChain è la rete "**Tor**".

Cos'è la rete Tor? - (<https://www.torproject.org>)

"**Tor** (acronimo di **T**e **O**nion **R**outer - in spagnolo) è un progetto il cui obiettivo principale è lo sviluppo di una rete di comunicazione distribuita a bassa latenza sovrapposta a Internet, in cui l'instradamento dei messaggi scambiati tra gli utenti non rivela la loro identità, cioè il loro indirizzo IP (anonimato a livello di rete) e che, inoltre, mantiene l'integrità e la segretezza delle informazioni che vi transitano".

Il secondo componente e non meno importante compito è quello di far sì che tutti i nodi di MiniBlocklyChain abbiano gli stessi dati in qualsiasi momento o che i loro database e file siano sincronizzati per eseguire questo compito tra i nodi sarà implementato "**syncthing**".

Cos'è la rete di sincronizzazione? - (<https://syncthing.net>)

Syncthing è un'applicazione gratuita di sincronizzazione di file peer-to-peer open source disponibile per Windows, Mac, Linux, Android, Solaris, Darwin e BSD. È possibile sincronizzare i file tra dispositivi su una rete locale o tra dispositivi remoti via Internet.

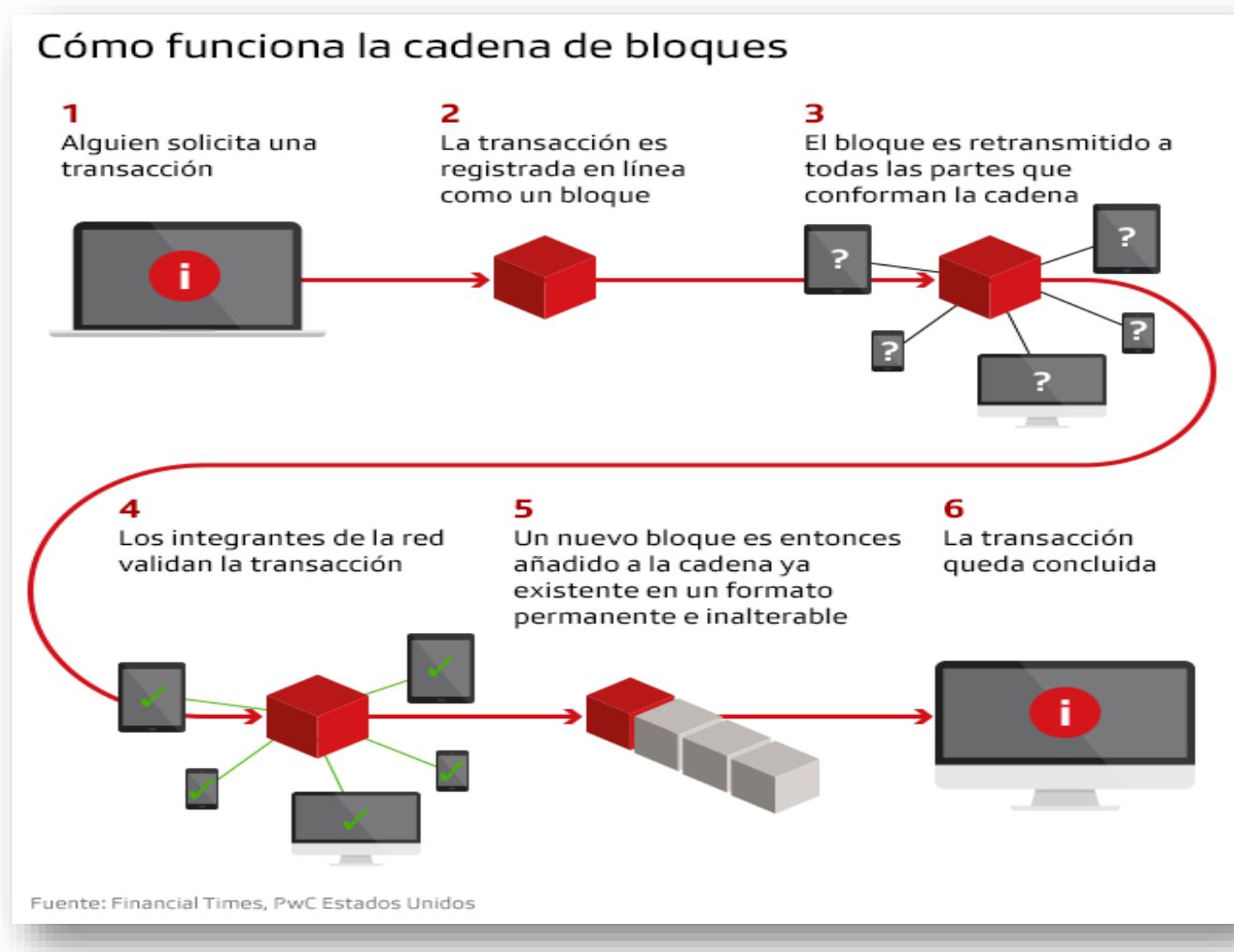
Sulla base delle due precedenti componenti di comunicazione possiamo iniziare a sviluppare una rete di fiducia tra i nodi e con il livello di sincronizzazione dei dati che sarà la spina dorsale o "core" dei processi aziendali per soddisfare ogni esigenza dell'utente o dell'azienda.

Il terzo componente della rete di comunicazione è il database Redis, che notificherà in tempo reale a tutti i nodi le nuove transazioni ricevute e le nuove transazioni inviate.

Cos'è la rete Redis DB? - (<https://redis.io>)

Redis è un motore di database in-memory, basato sulla memorizzazione in tabelle di hash (chiave/valore), ma che può essere utilizzato come database durevole o persistente.

7. Diagramma di funzionalità BlocklyChain (Mini BlocklyChain).



8. Cos'è il Mini BlocklyCode?

I Mini BlocklyCode sono programmi creati in linguaggio Java e sono memorizzati in un repository indipendente dai nodi, sono normalmente denominati come contratti intelligenti, questi programmi sono progettati con condizioni logiche in modo che quando queste condizioni sono soddisfatte vengono eseguiti automaticamente senza dipendere da alcuna autorizzazione o intervento umano esterno.

"Un contratto intelligente è un programma per computer che facilita, assicura, fa rispettare ed esegue gli accordi registrati tra due o più parti (ad esempio, individui o organizzazioni). In quanto tali, essi li aiuterebbero a negoziare e a definire tali accordi che determineranno determinate azioni a seguito del rispetto di una serie di condizioni specifiche."

Un contratto intelligente è un programma che vive in un sistema non controllato da una delle due parti, o dai loro agenti, e che esegue un contratto automatico che funziona come una frase "if-then" di qualsiasi altro programma per computer. Con la differenza che si fa in un modo che interagisce con i beni reali. Quando si innesca una condizione pre-programmata, non soggetta ad alcun giudizio umano, il contratto intelligente esegue la corrispondente clausola contrattuale.

Essi mirano a fornire una maggiore sicurezza rispetto al diritto contrattuale tradizionale e a ridurre i costi di transazione associati alla stipula di contratti. Il trasferimento del valore digitale attraverso un sistema che non richiede fiducia (ad es. bitcoins) apre la porta a nuove applicazioni che possono avvalersi di contratti intelligenti. "

Mini BlocklyCode è rivolto a sviluppatori con esperienza nella programmazione java. In Mini BlocklyChain alcuni tipi di biblioteche sono limitati per la sicurezza dello stesso sistema, tuttavia, le biblioteche per creare transazioni per inviare o ricevere un certo valore contrattuale creato o concordato da due o più parti possono essere create.

Ogni Mini BlocklyChain è conforme alle seguenti premesse:

- ✓ Qualsiasi Mini BlocklyChain creato è basato sull'esecuzione di soli messaggi e non di esecuzioni sul sistema, tuttavia, questi messaggi possono contenere autorizzazioni, approvazioni di contratti fisici o azioni fisiche.
- ✓ Ogni Mini BlocklyChain creato deve passare attraverso il blocco di comunicazione "auditor", che ha il compito di monitorare il codice maligno o il codice proibito per rivedere le frasi o gli ordini non autorizzati nel sistema.
- ✓ Tutti i Mini BlocklyChain vengono eseguiti solo sulla JVM del nodo sorgente, i programmi non vengono eseguiti globalmente per la protezione del sistema.

Per maggiori dettagli vedere l'appendice "Mini BlocklyChain per sviluppatori".

9. Installazione della rete di comunicazione Mini BlocklyChain

1. Installazione e configurazione della rete Mini SQLSync

La rete Mini SQLSync è incaricata di ricevere le transazioni dai nodi in modo da poterle elaborare. Questa rete è formata da una rete principale che è attraverso "Peer to Peer" tra i nodi e da una rete di backup chiamata Mini Sentinel RESTful.

Inizieremo con l'installazione della rete di backup RESTful Mini Sentinel, questo servizio è quello che riceverà le transazioni dai nodi, questo servizio si basa sulla memorizzazione delle transazioni per un determinato periodo di tempo per poi convertire le informazioni attraverso un connettore che inietterà una coda di tutte le transazioni criptate in un servizio Redis. Successivamente il servizio di database Redis eseguirà in tempo reale il rilascio della coda delle transazioni criptate a tutti i nodi che integrano la rete Mini BlocklyChain.

Un servizio di tipo RESTful o design è un modo semplice e facile per consultare, modificare, cancellare e/o inserire informazioni in un database attraverso un URL o un indirizzo internet, nel nostro caso useremo il database SQLite per le sue caratteristiche di essere tutte le informazioni in un unico file. Per un uso di requisiti con esigenze di scalatura maggiore saremo in grado di utilizzare un altro tipo di database come MySQL, Oracle, DB2 o un altro database commerciale, semplicemente dovremo cambiare il connettore di Mini BloclyChain di SQLite (versione gratuita) per il connettore di Mini BlocklyChain DB altri (versione commerciale).

NOTA IMPORTANTE: La configurazione RESTful Mini Sentinel non elabora alcun tipo di transazione in qualsiasi momento, è solo il servizio che formatta "modella l'informazione" in modo che i nodi possano eseguire l'elaborazione della transazione in modo corretto e in un tempo pianificato e sincronizzato per tutti i nodi.

L'installazione del servizio RESTful è basata su un database SQLite. Questa installazione sarà effettuata in ambiente Windows per scopi pratici, tuttavia, questo modello può essere facilmente replicato in un sistema operativo di tipo Linux.

Per coloro che desiderano verificare le prestazioni e il supporto delle query e degli inserimenti SQLite, si prega di fare riferimento a questi test di prestazione:

<https://www.sami-lehtinen.net/blog/sqlite3-performance-testing>

<https://sqlite.org/src4/doc/trunk/www/lsmperf.wiki>

Installazione della rete di backup RESTful Mini Sentinel. Installeremo e configureremo questo servizio in un computer Windows 10, tuttavia, il processo è stato installato facilmente anche in un ambiente Ubuntu 18.09 Linux.

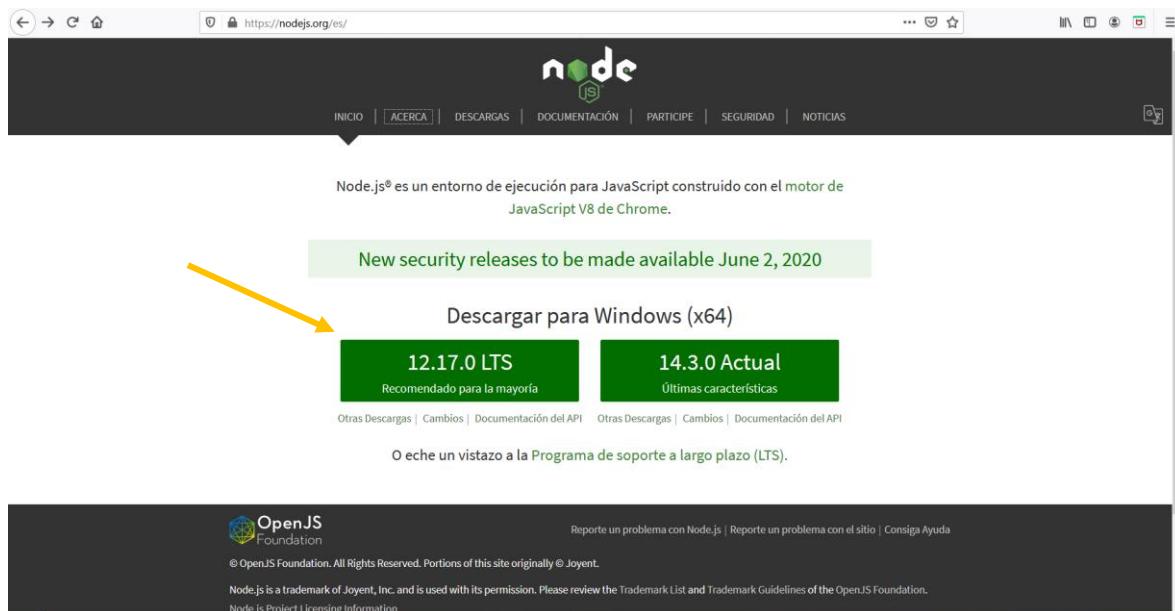
Requisiti:

- ✓ Server di database SQLite in modalità RESTful (<https://github.com/olsonpm/sqlite-to-rest>).
- ✓ Connettore Sentinel SQLite-Redis Sentinel Connector
- ✓ Redis server di database in modalità Master-Slave (<https://redis.io/topics/replication>)

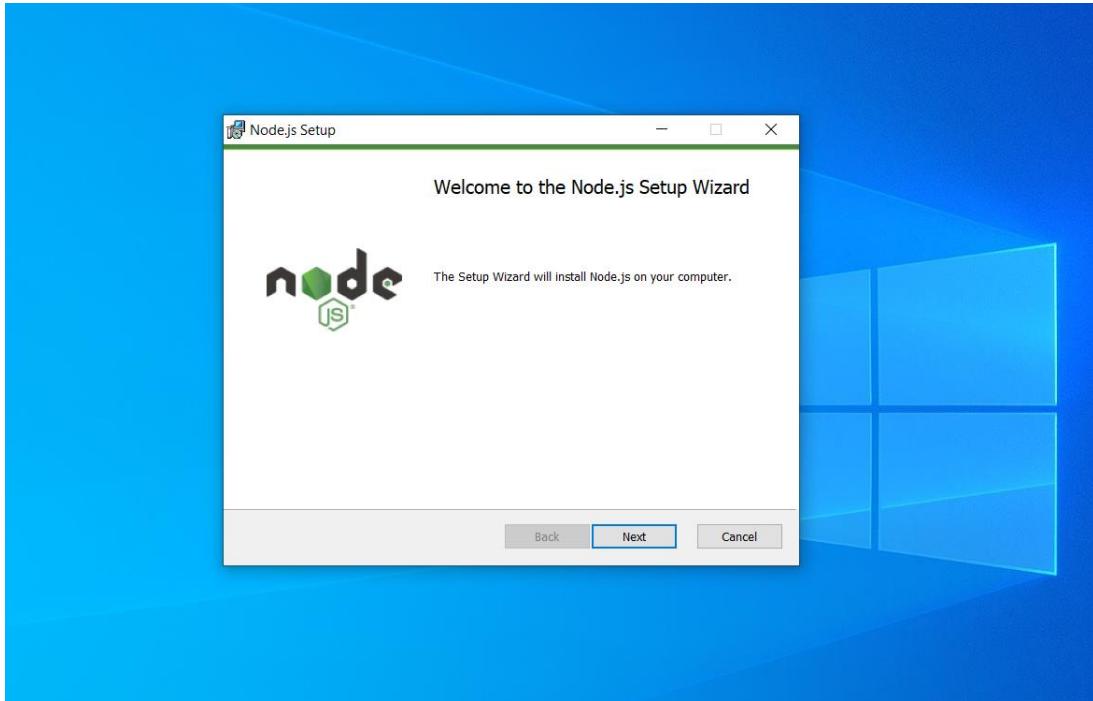
Installazione del server di database SQLite in modalità RESTful

Per l'installazione dobbiamo iniziare con i seguenti pacchetti software, Nodejs, sqlite3 e Git Bash per Windows.

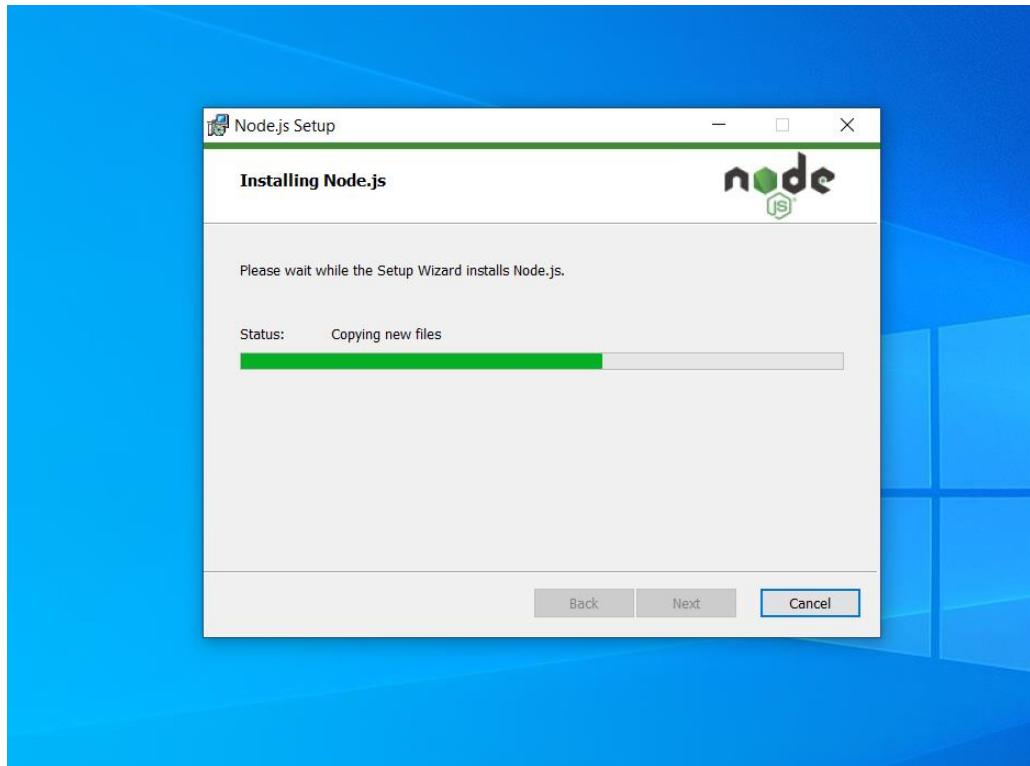
Per ottenere i Nodejs abbiamo consultato il loro sito ufficiale <https://nodejs.org/es/> e abbiamo scelto la versione "Recommended for Most".

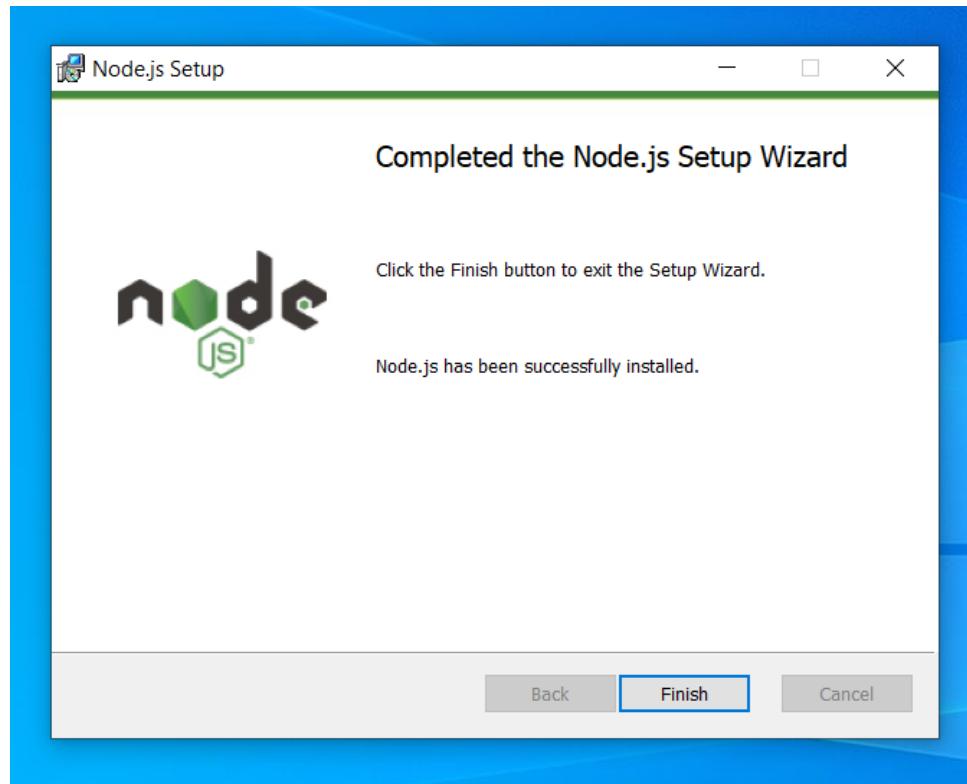


Dopo aver scaricato il file con estensione .msi facciamo doppio click per installarlo.



Eseguiamo l'installazione di default, basta cliccare su "Avanti" senza scegliere le opzioni aggiuntive richieste.



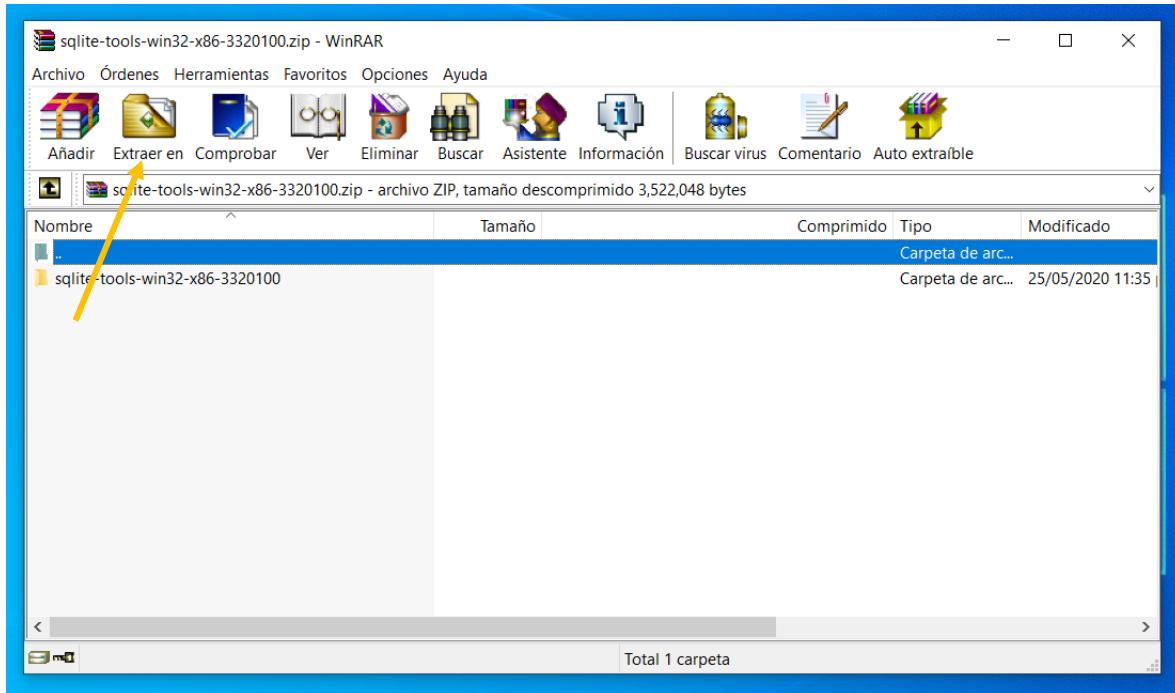


Da quando abbiamo finito l'installazione di Nodejs continuiamo con l'installazione del database SQLite.

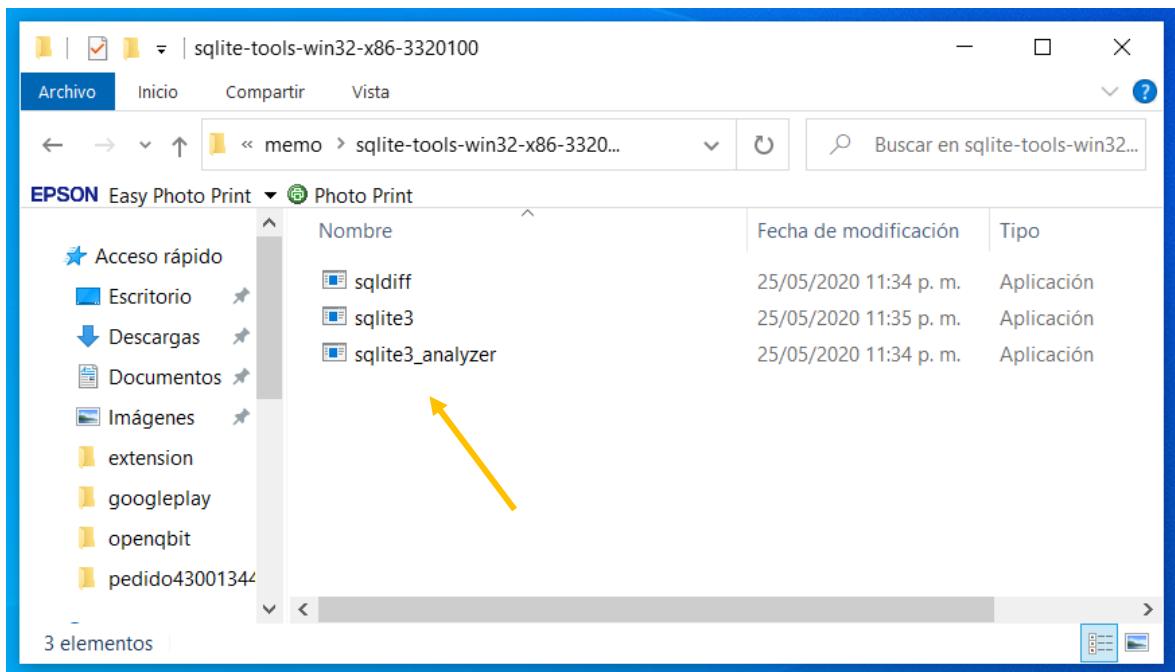
Andiamo sul loro sito ufficiale, <https://www.sqlite.org/index.html> e clicchiamo sulla parte dove si "scarica".

A screenshot of the SQLite.org homepage. The header shows the SQLite logo and navigation links: Home, About, Documentation, Download, License, Support, Purchase. A yellow arrow points to the "Download" link. Below the header, there's a section titled "What Is SQLite?" with a brief description and a "More Information..." link. Another section discusses the SQLite file format and its widespread use. At the bottom, there's a link to the source code and information about the latest release (Version 3.32.1).

Poi scegliamo l'opzione dove dice "**Precompiled Binaries for Windows**" scegliamo e clicchiamo sulla versione del software "**sqlite-tools-win32-x86-3320100.zip**" quando finiamo il download sul nostro computer procediamo a decomprimere il file questo lo facciamo in modo semplice, apriamo la cartella dove il file è stato scaricato e diamo un doppio click al file per decomprimere.

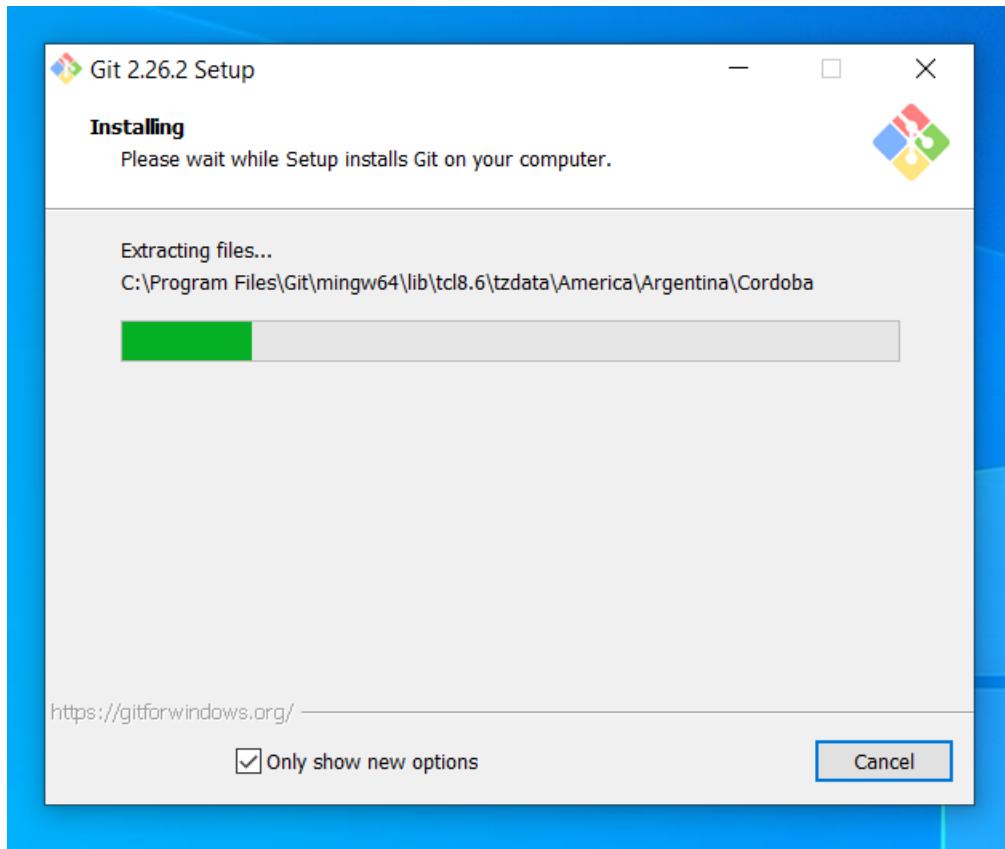


Dopo averlo decompresso avremo tre file, questi sono il nostro ambiente per creare il nostro database SQLite che useremo in seguito.



Inizieremo installando Git Bash, un emulatore di terminale simile a Linux che ci aiuterà a far funzionare correttamente il servizio di backup RESTful.

Lo scaricheremo dal loro sito ufficiale, <https://gitforwindows.org/> e procederemo all'installazione con le opzioni predefinite che indica.



Ora che abbiamo installato nodejs, sqlite e Git Bash sulla nostra macchina Windows 10 procediamo ad installare l'ambiente che supporterà il nostro database SQLite in modalità RESTful.

Al momento del download del software che darà le funzionalità di RESTful a SQLite. Per questo dobbiamo andare al seguente sito, <https://github.com/olsonpm/sqlite-to-rest> in questo cliccheremo sul pulsante verde a destra "Clona o scarica" e sceglieremo l'opzione "Download ZIP" questo scaricherà il software nel nostro computer.

Mini BlocklyChain - fai da te - "Do It Yourself"

No description or website provided.

automatic-api

Branch: dev New pull request

Find file Clone or download ▾

Clone with HTTPS ⓘ
Use Git or checkout with SVN using the web URL.
<https://github.com/olsonpm/sqlite-to-rest>

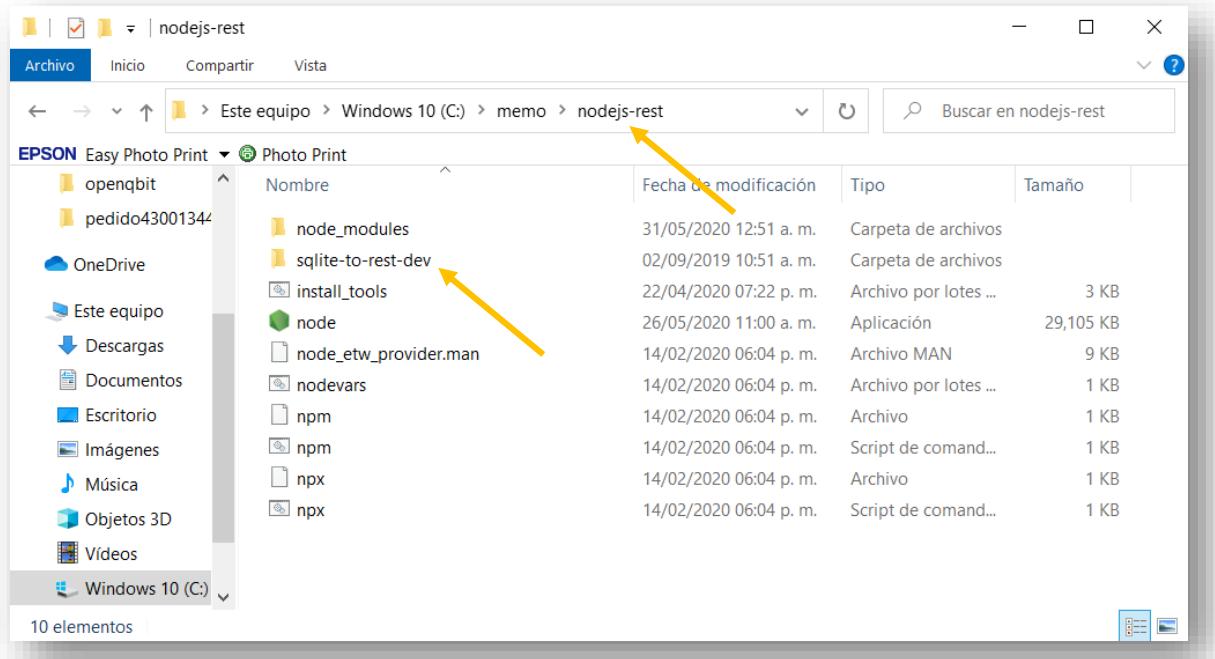
Open in Desktop Download ZIP

9 months ago 9 months ago 9 months ago

File	Description	Modified
bin	add prettier and eslint	9 months ago
cli/commands	add prettier and eslint	9 months ago
docs	add prettier and eslint	9 months ago
lib	add prettier and eslint	9 months ago
tests	add prettier and eslint	9 months ago
.gitignore	add prettier and eslint	9 months ago

Dopo averlo scaricato nel nostro computer procediamo a decomprimere all'interno della directory o della cartella dove abbiamo installato il programma **nodejs** e avremo una directory con il nome "**sqlite-to-rest-dev**".

NOTA: È importante che la directory **sqlite-to-rest-dev** sia all'interno della directory in cui è stato installato **nodejs**.



Avendo tutto installato procediamo con la configurazione del database SQLite che useremo per memorizzare le transazioni dei nodi nell'ambiente di backup RESTful.

Progettazione della tabella e struttura dei dati. Comandi da eseguire online nella lista di comandi CMD

sqlite3 op.sqlite3 "CREATE TABLE trans (id integer primary key, addro, addrd, value);".

sqlite3 op.sqlite3 "CREATE TABLE sign (id integer primary key, riferimenti trans_id trans (id), sign, hash);".

```

Símbolo del sistema
C:\memo\sqlite-tools-win32-x86-3320100>sqlite3 op.sqlite3 "CREATE TABLE trans(id integer primary key, addro, addrd, value);"
C:\memo\sqlite-tools-win32-x86-3320100>sqlite3 op.sqlite3 "CREATE TABLE sign(id integer primary key, trans_id references trans (id), sign, hash);"
C:\memo\sqlite-tools-win32-x86-3320100>dir
El volumen de la unidad C es Windows 10
El número de serie del volumen es: E019-5C05

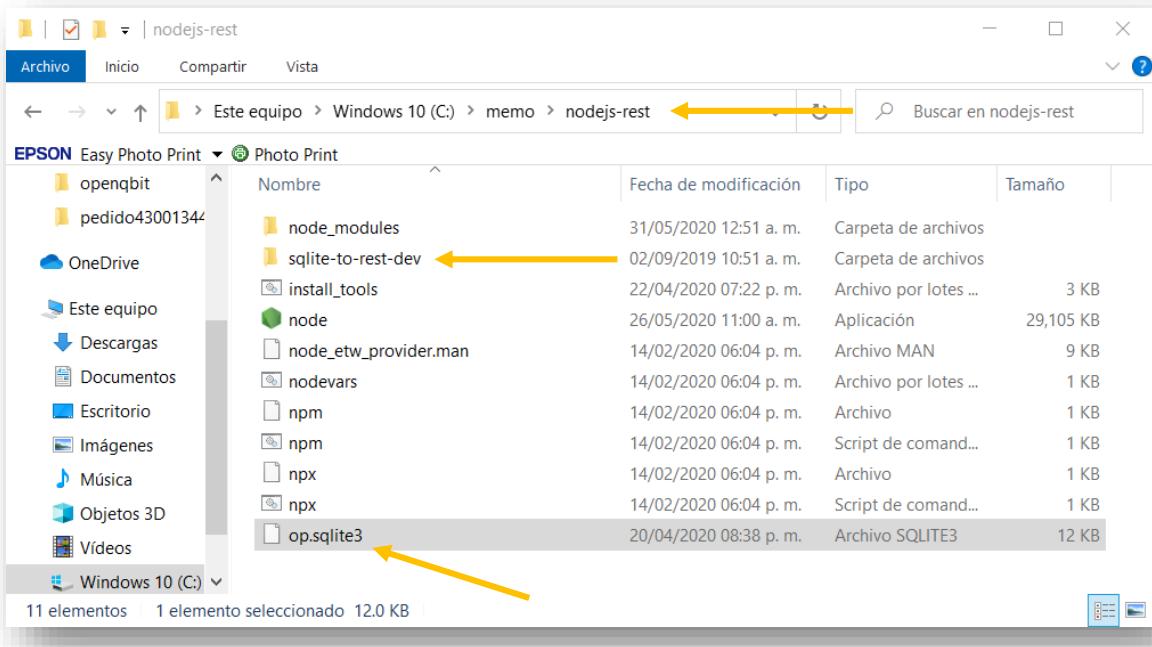
Directorio de C:\memo\sqlite-tools-win32-x86-3320100

31/05/2020 02:29 a. m.    <DIR>        .
31/05/2020 02:29 a. m.    <DIR>        ..
31/05/2020 02:29 a. m.        12,288 op.sqlite3
25/05/2020 11:34 p. m.      516,608 sqldiff.exe
25/05/2020 11:35 p. m.      972,800 sqlite3.exe
25/05/2020 11:34 p. m.      2,032,640 sqlite3_analyzer.exe
              4 archivos       3,534,336 bytes
              2 dirs   137,272,078,336 bytes libres

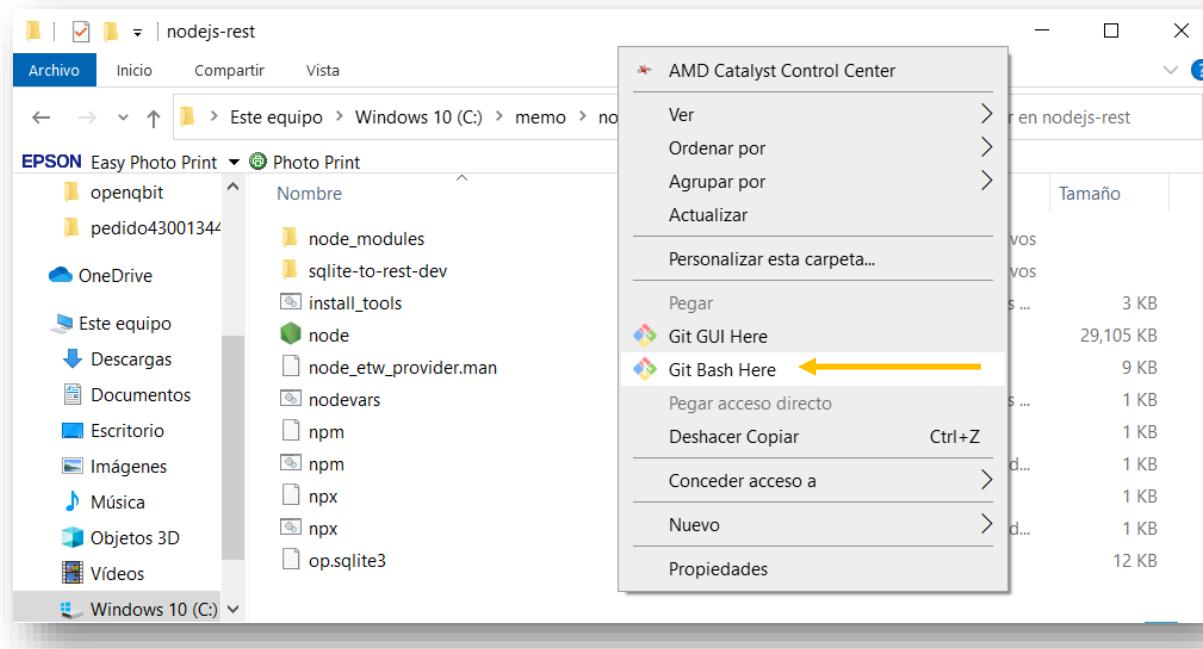
C:\memo\sqlite-tools-win32-x86-3320100>

```

Dopo aver creato il database op.sqlite3 dobbiamo fare una copia nella directory dove è stato installato il **nodejs**. Nella directory **nodejs** deve essere presente anche la copia di "**sqlite-to-rest-dev**".



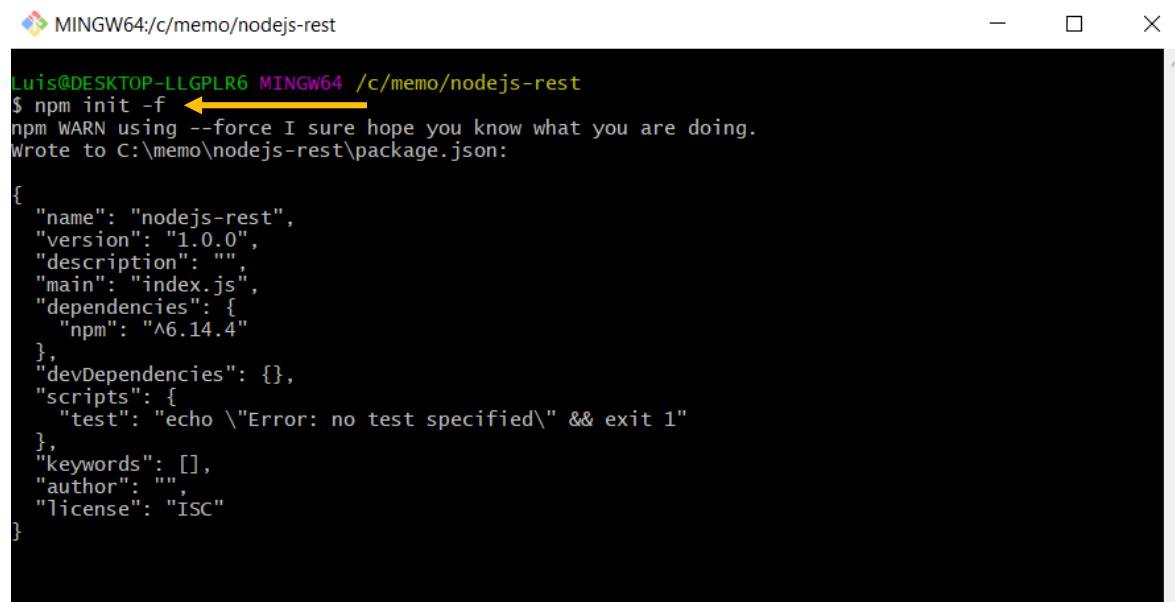
Ci mettiamo nella cartella dove si trova l'installazione dei **nodejs** e dove dovrebbe trovarsi anche il software "**sqlite-to-rest-dev**". Puntare la cartella con il puntatore e cliccare con il



tasto destro del mouse per visualizzare il menu e scegliere dove dice "Git Bash" per aprire un terminale.

Nel nuovo terminale Git Bash aperto, eseguiamo i seguenti comandi:

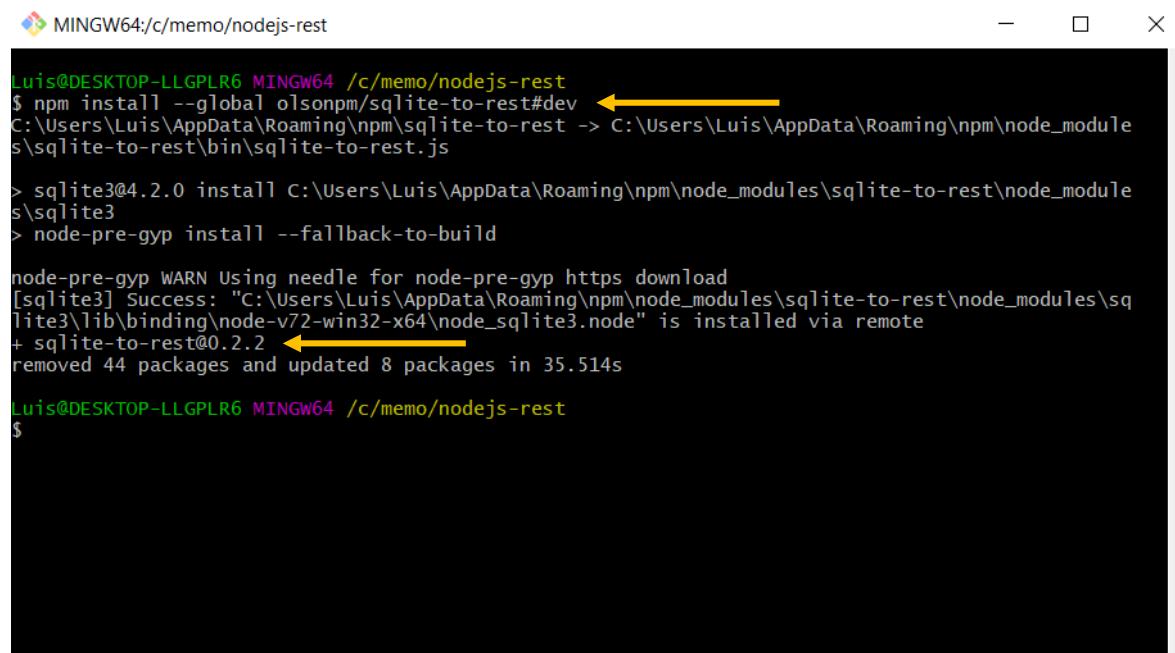
\$ npm init -f



```
Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$ npm init -f ←
npm WARN using --force I sure hope you know what you are doing.
Wrote to C:\memo\nodejs-rest\package.json:

{
  "name": "nodejs-rest",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "dependencies": {
    "npm": "^6.14.4"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

\$ npm install --global olsonpm/sqlite-to-rest#dev



```
Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$ npm install --global olsonpm/sqlite-to-rest#dev ←
C:\Users\Luis\AppData\Roaming\npm\sqlite-to-rest -> C:\Users\Luis\AppData\Roaming\npm\node_modules\sqlite-to-rest\bin\sqlite-to-rest.js

> sqlite3@4.2.0 install C:\Users\Luis\AppData\Roaming\npm\node_modules\sqlite-to-rest\node_modules\sqlite3
> node-pre-gyp install --fallback-to-build

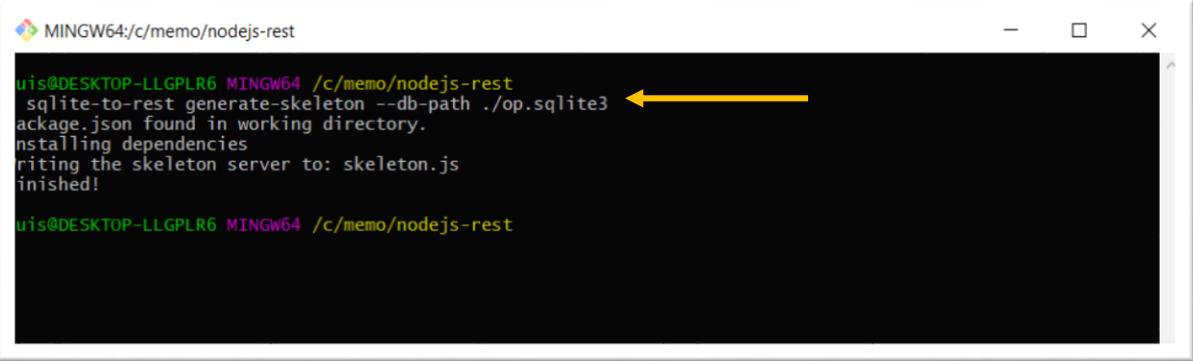
node-pre-gyp WARN Using needle for node-pre-gyp https download
[sqlite3] Success: "C:\Users\Luis\AppData\Roaming\npm\node_modules\sqlite-to-rest\node_modules\sqlite3\lib\binding\node-v72-win32-x64\node_sqlite3.node" is installed via remote
+ sqlite-to-rest@0.2.2 ←
removed 44 packages and updated 8 packages in 35.514s

Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$
```

Dopo l'esecuzione del comando, apparirà l'installazione del pacchetto "sqlite-to-rest".

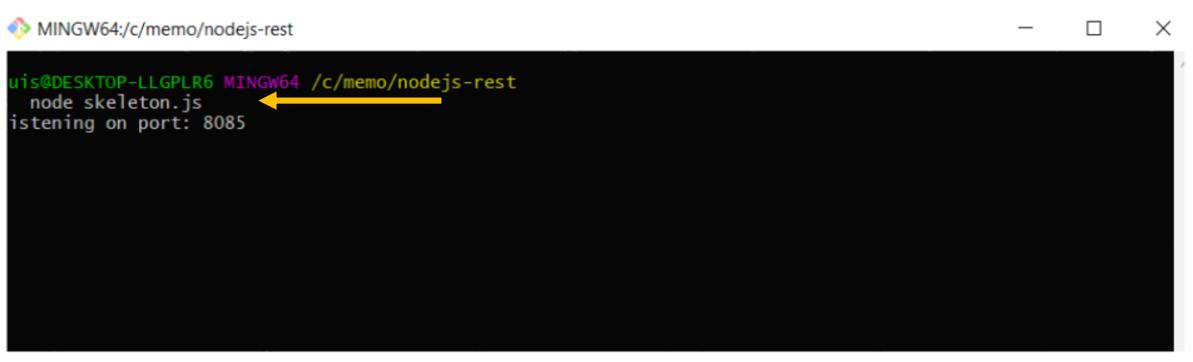
Abbiamo generato l'ambiente RESTful per SQLite con la base **op.sqlite3** che abbiamo creato in precedenza.

Eseguiamo il comando: **\$ sqlite-to-rest generatore-scheletro --db-path ./op.sqlite3**



```
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
sqlite-to-rest generate-skeleton --db-path ./op.sqlite3 ←
package.json found in working directory.
Installing dependencies
Writing the skeleton server to: skeleton.js
finished!
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
```

Abbiamo avviato il servizio RESTful SQLite sulla porta di default 8085. Eseguiamo il seguente comando: **\$ node skeleton.js**



```
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
node skeleton.js ←
listening on port: 8085
```

Abbiamo testato il servizio RESTful con l'aggiunta di dati e l'interrogazione di dati.

The screenshot shows a terminal window titled 'MINGW64/c/memo/nodejs-rest'. It contains the following text:

```
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
curl -s -H "Content-Type: application/json" -d '{"addr": "QWERTY1234", "addrd": "ASDFG4567", "value": "999"}' http://localhost:8085/trans
"id":6,"addr": "QWERTY1234","addrd": "ASDFG4567","value": "999"}" id":6,"addr": "QWERTY1234","addrd": "ASDFG4567","value": "999"}'
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
curl -s http://localhost:8085/trans
```

Two yellow arrows point to the command 'curl -s http://localhost:8085/trans' and its response. The response is a JSON array of objects representing transactions:

```
[{"id":1,"addr": "CO","addrd": "Boulder","value": "Avery"}, {"id":2,"addr": "WI","addrd": "New Glarus","value": "New Glarus"}, {"id":3,"addr": "WI","addrd": "Madison","value": "One Barrel"}, {"id":4,"addr": "WI","addrd": "Madison","value": "One Barrel"}, {"id":5,"addr": "WI","addrd": "Madison","value": "One Barrel"}, {"id":6,"addr": "QWERTY1234","addrd": "ASDFG4567","value": "999"}]
```

Per testare il servizio SQLite RESTful utilizziamo i seguenti comandi:

Per inserire i dati nella tabella trans che si trova all'interno del database op.sqlite3:

```
$ curl -s -H "Content-Type: application/json" -d '{"addr": "QWERTY1234", "addrd": "ASDFG4567", "value": "999"}' http://localhost:8085/trans
```

Per effettuare una ricerca di tutti i dati della tabella dei trans:

```
$ curl -s -H 'range: rows=0-2' http://localhost:8085/trans
```

Per verificare come utilizzare in dettaglio tutti i servizi abilitati RESTful (interrogare, inserire, aggiornare e/o cancellare i dati) consultare l'[appendice "Comandi Restful SQLite GET/POST"](#).

NOTA: Nella precedente installazione, tutte le query sono state effettuate nell'URL con l'indirizzo "localhost", ma quando il server è esposto con un IP pubblico (internet) o privato (Wifi) funzionerà senza alcun problema. Lo testeremo durante i test di comunicazione tra il servizio SQLite RESTful e i nodi della rete di comunicazione che formano Mini BlocklyChain.

Installazione del database Redis per il servizio di rete di backup, per scaricare il software redis per Windows dobbiamo andare sul sito, <https://github.com/microsoftarchive/redis/releases/tag/win-3.2.100> e scegliere il pacchetto ZIP.

3.2.100

enricogior released this on 1 Jul 2016 · 1210 commits to 3.0 since this release

This is the first release of Redis on Windows 3.2.

This release is based on antirez/redis/3.2.1 plus some Windows specific fixes. It has passed all the standard tests but it hasn't been tested in a production environment.

Therefore, before considering using this release in production, make sure to test it thoroughly in your own test environment.

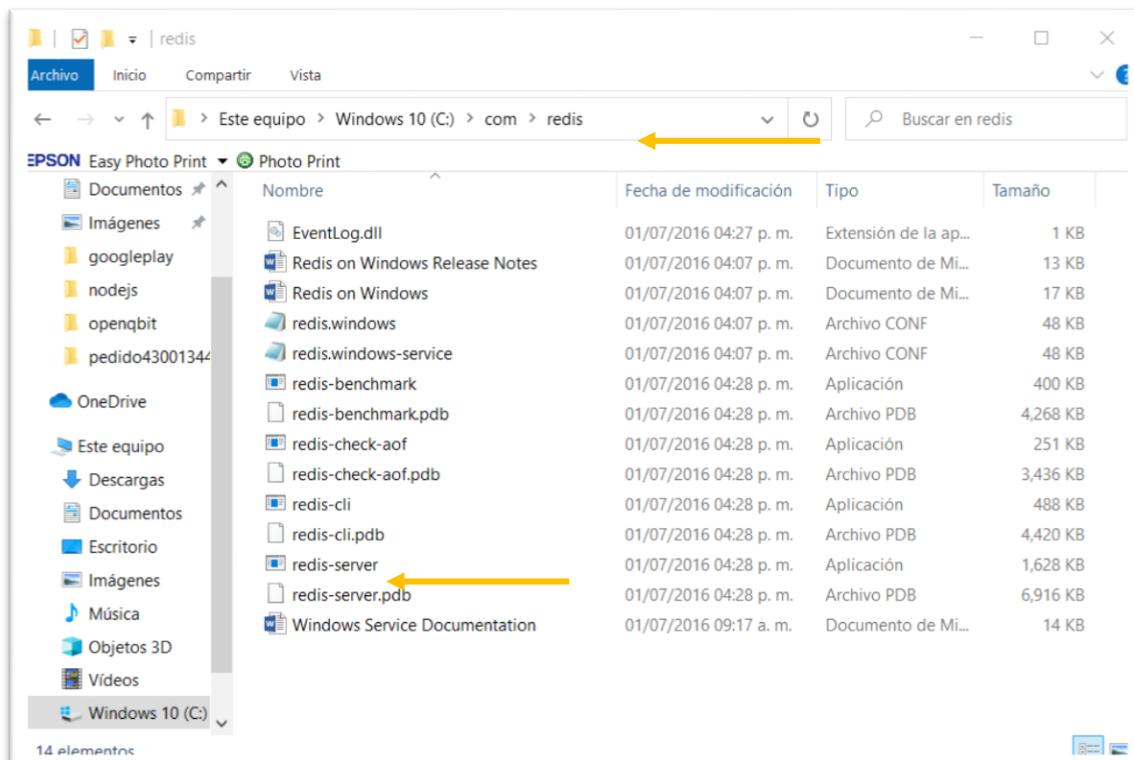
See the [release notes](#) for details.

Assets 4

Redis-x64-3.2.100.msi	5.8 MB
Redis-x64-3.2.100.zip	4.98 MB
Source code (zip)	
Source code (tar.gz)	

Per localizzare l'installazione creeremo una directory chiamata "redis" all'interno di Windows e scaricheremo il file con estensione ZIP, lo decomprimiamo nella directory creata in precedenza, abbiamo già l'installazione completata redis.

Testiamo l'installazione eseguendo il server con un doppio clic sul comando "**redis-server**".



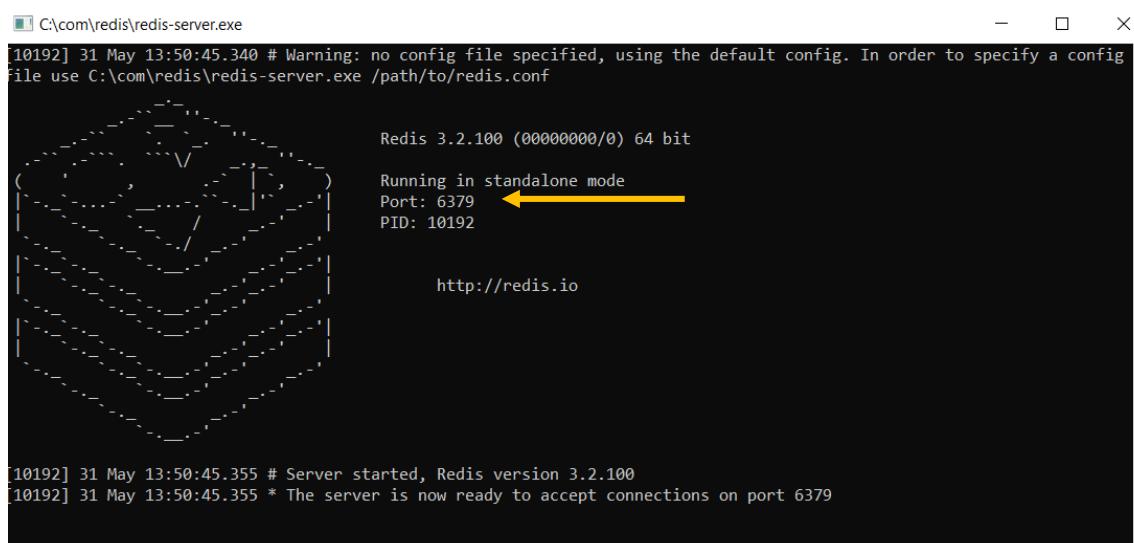
Dopo aver eseguito il comando "redis-server" vedremo il server girare in un terminale di comando CMD di Windows sulla porta predefinita 6379:

In questo test abbiamo una versione di Redis 3.2.100 per Windows 10, nel caso del sistema operativo Linux abbiamo la versione 6.0.4 (rilasciata a maggio 2020) tuttavia per la nostra configurazione Mini BlocklyChain la versione Windows ha abbastanza caratteristiche di funzionalità di cui abbiamo bisogno.

Per fermare l'esecuzione si effettua la combinazione di tasti Ctrl + C. Si procede alla configurazione del database Redis 3.2.100 per Windows 10 con una configurazione tra Redis e i nodi di rete di comunicazione Mini SQLSync, tipo **Master(Master)-Slave(Slave)** è distribuito come segue:

Il Master è il server Redis per Windows 10 che stiamo configurando e che replicherà i dati in tempo reale e sincronizzato con le stesse informazioni a tutti i nodi (telefoni cellulari). Questi nodi avranno un server Redis installato in modalità **Slave**.

A questo punto iniziamo a vedere come funziona la rete di backup che stiamo installando e configurando. Questa configurazione ci servirà per comunicare con tutti i nodi in tempo reale, ci aiuterà a comunicare a tutti i nodi della rete quando ci sarà una nuova coda di transazioni da elaborare da parte dei nodi, in modo che ogni nodo abbia la stessa probabilità di poter elaborare le informazioni della "coda di transazioni".



```
C:\com\redis>redis-server.exe
[10192] 31 May 13:50:45.340 # Warning: no config file specified, using the default config. In order to specify a config
file use C:\com\redis\redis-server.exe /path/to/redis.conf

Redis 3.2.100 (00000000/0) 64 bit
Running in standalone mode
Port: 6379 ←
PID: 10192

http://redis.io

[10192] 31 May 13:50:45.355 # Server started, Redis version 3.2.100
[10192] 31 May 13:50:45.355 * The server is now ready to accept connections on port 6379
```

Iniziamo la configurazione del connettore **SQLite-Redis Sentinel**.

Questo connettore è un programma sviluppato in linguaggio Java e come indica il suo nome collega i database SQLite e Redis (**Master**).

La funzione del connettore è quella di trasmettere le informazioni (transazioni) da SQLite a Redis (**Master**) e questo invia la "coda delle transazioni" ai nodi (**Slave**).

Per maggiori dettagli sul codice Java del connettore **SQLite-Redis Sentinel si veda** l'allegato "SQLite-Redis Java Code Connector".

Configurazione del file **redis.conf** del database Redis (**Master**) per Windows 10.

Aggiungere le seguenti modifiche o direttive al file, salvare le modifiche e avviare il server Redis

Iniziate trovando l'impostazione **tcp-keepalive** e impostatela su 60 secondi come suggerito dai commenti. Questo aiuterà Redis a rilevare i problemi di rete o di servizio:

tcp-keepalive 60

Trovate la direttiva **requirepass** e configuratela con una forte passphrase. Mentre il vostro traffico Redis deve essere protetto da terzi, questo fornisce l'autenticazione a Redis. Poiché Redis è veloce e non valuta i tentativi di passphrase borderline, scegliete una passphrase forte e complessa per proteggere dai tentativi di forza bruta:

requirepass type_your_network_master_password

esempio:

requirepass FPqwedsLMdf76ass7asddf2g45vBN8ty99

Infine, ci sono alcune impostazioni opzionali che si possono regolare a seconda dello scenario di utilizzo.

Se non si desidera che Redis inserisca automaticamente le chiavi più vecchie e meno usate mentre si riempie, è possibile disattivare la rimozione automatica delle chiavi:

maxmemoria-policy noevasion

Per una maggiore garanzia di durata, è possibile abilitare la persistenza dei file add-on-only. Questo aiuterà a ridurre al minimo la perdita di dati in caso di guasto del sistema a scapito di file più grandi e di prestazioni leggermente più lente:

Appendonamente sì

appendfilename "redis-staging-ao.aof".

Procedere per salvare le modifiche e riavviare il servizio Redis per Windows 10, fermarsi con i tasti Ctrl + C ed eseguire nuovamente la riga di comando CMD di Windows:

C:\redis_redis_directory redis_server

Nel nostro esempio possiamo vedere che abbiamo un nodo (**Slave**) collegato.

```

Símbolo del sistema
:\\com\\redis> redis-server redis.conf
1:C 31 May 2020 23:44:56.633 # o000o000o000 Redis is starting o000o000o000
1:C 31 May 2020 23:44:56.634 # Redis version=5.0.7, bits=64, commit=00000000, modified=0,
id=51, just started
1:C 31 May 2020 23:44:56.634 # Configuration loaded
1:M 31 May 2020 23:44:56.635 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 3.2.100 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 51

http://redis.io

1:M 31 May 2020 23:44:56.648 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.51:M 31 May 2020 23:44:56.648 # Server initialized
1:M 31 May 2020 23:44:56.649 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
1:M 31 May 2020 23:44:56.669 * DB loaded from append only file: 0.020 seconds
1:M 31 May 2020 23:44:56.669 * Ready to accept connections
1:M 31 May 2020 23:49:57.013 * 10 changes in 300 seconds. Saving...
1:M 31 May 2020 23:49:57.031 * Background saving started by pid 82
2:C 31 May 2020 23:49:57.052 * DB saved on disk
1:M 31 May 2020 23:49:57.133 * Background saving terminated with success
1:M 31 May 2020 23:50:24.600 * Replica 192.168.1.68:6379 asks for synchronization
1:M 31 May 2020 23:50:24.602 * Full resync requested by replica 192.168.1.68:6379
1:M 31 May 2020 23:50:24.602 * Starting BGSAVE for SYNC with target: disk
1:M 31 May 2020 23:50:24.619 * Background saving started by pid 83
3:C 31 May 2020 23:50:24.642 * DB saved on disk
1:M 31 May 2020 23:50:24.670 * Background saving terminated with success
1:M 31 May 2020 23:50:24.689 * Synchronization with replica 192.168.1.68:6379 succeeded

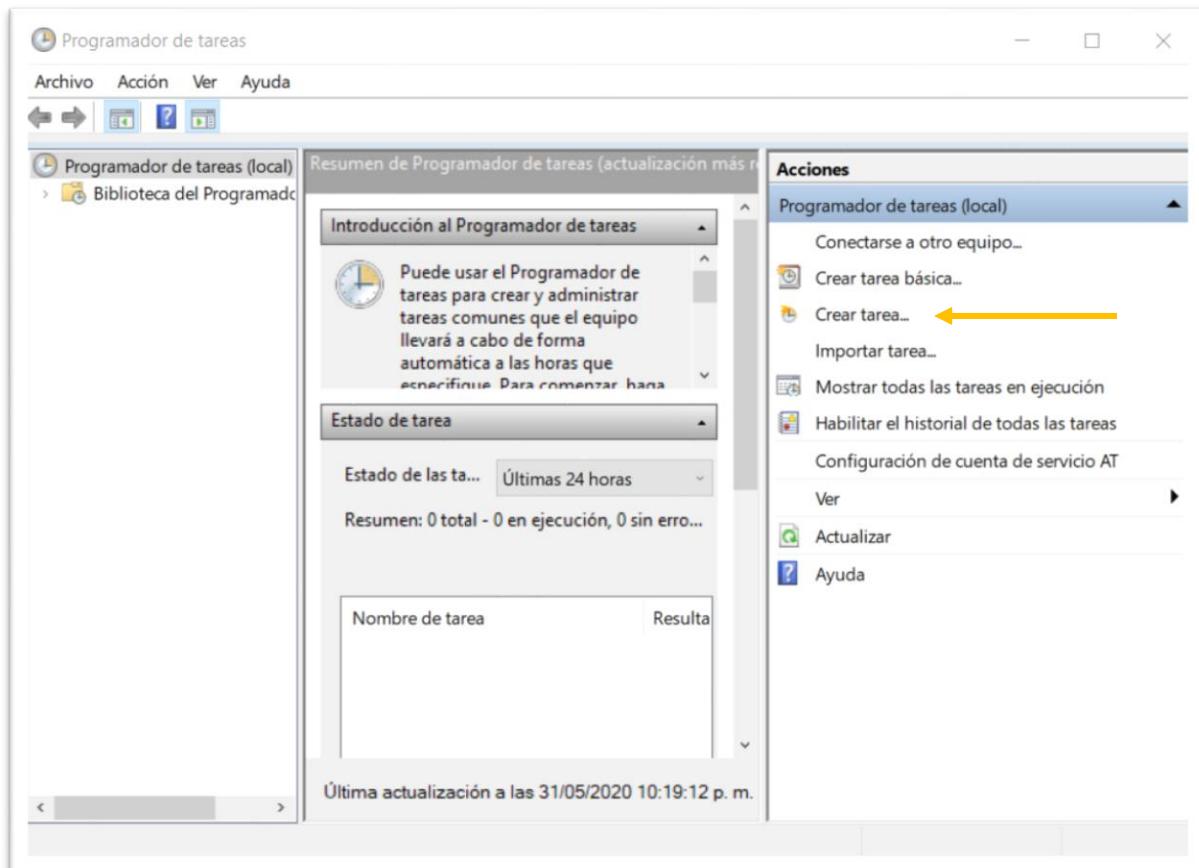
```

Possiamo vedere che abbiamo sincronizzato un nodo con l'IP 192.168.1.68 nella porta predefinita 6379.

Ora dobbiamo programmare nel sistema operativo Window 10 il compito di eseguire automaticamente il connettore **SQLite-Redis Sentinel**. Lo facciamo con lo strumento di Windows 10 è lo eseguiamo dal basso a sinistra digitando "Task Scheduler".



Creeremo un nuovo task "Crea task" dove includeremo l'esecuzione del connettore.

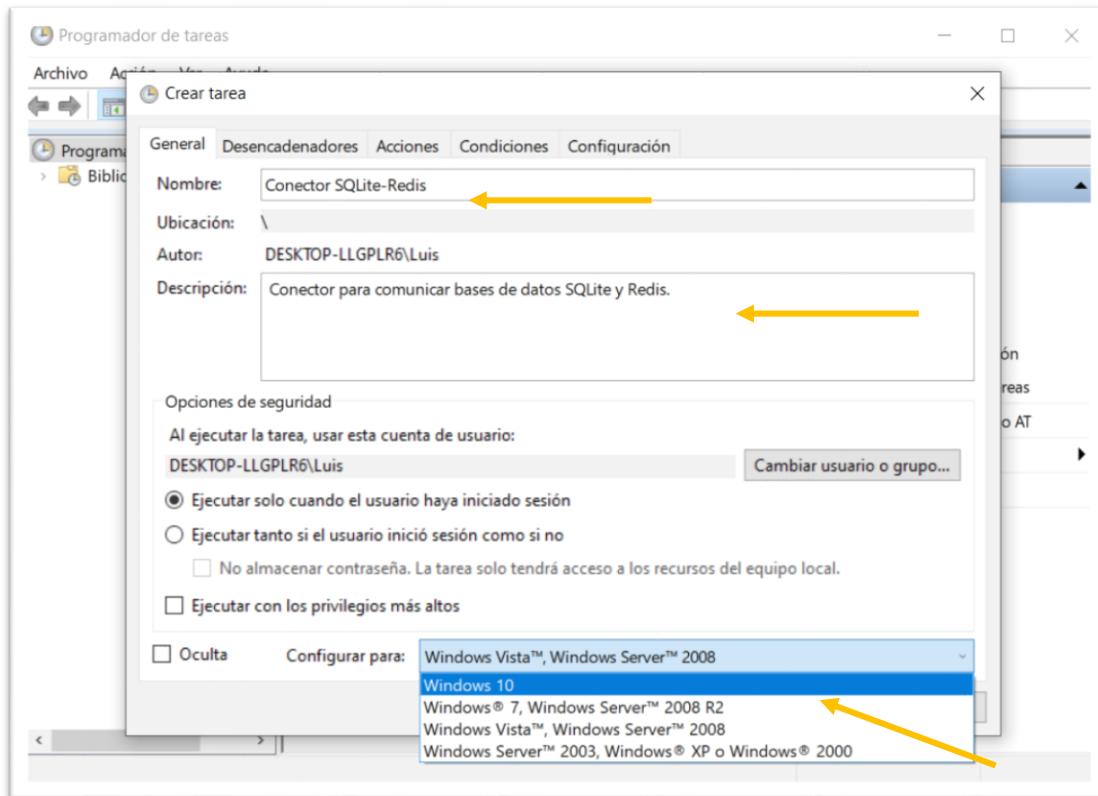


Cliccando su "Crea attività" si aprirà un'ulteriore finestra dove dovremo fornire i seguenti parametri nella scheda "Generale":

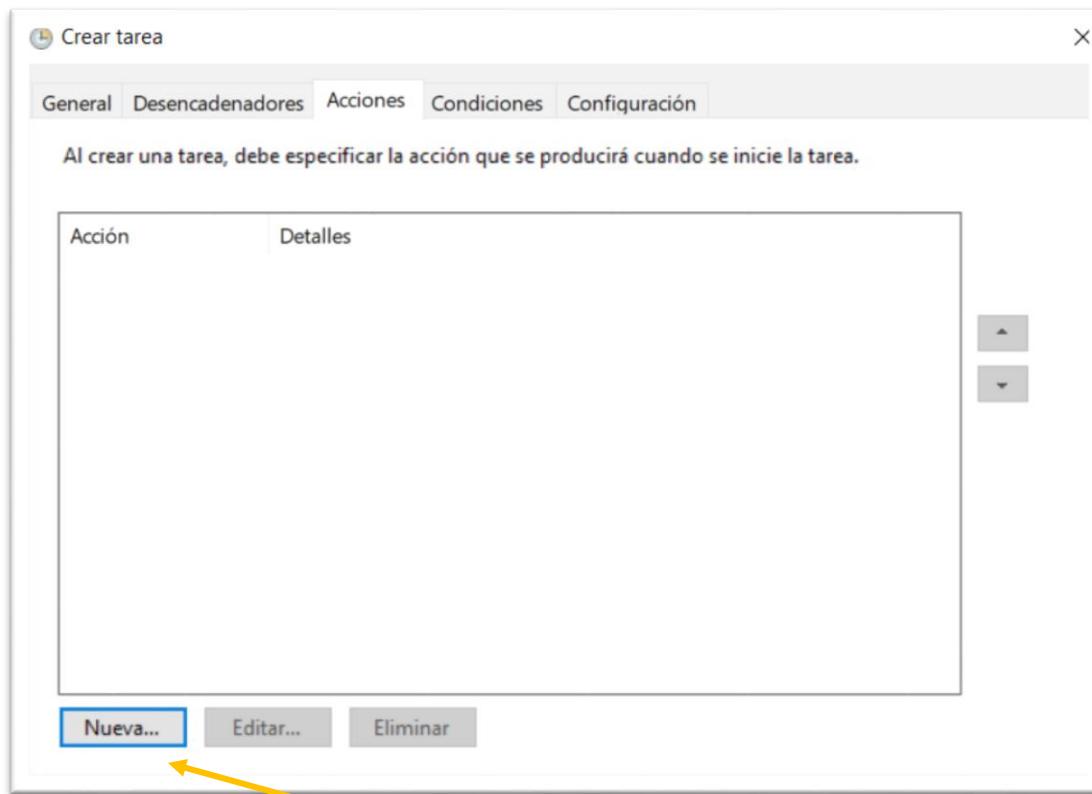
Nome: task_name

Descrizione: opzionale

Configurare per: select_operating_system_windows_version



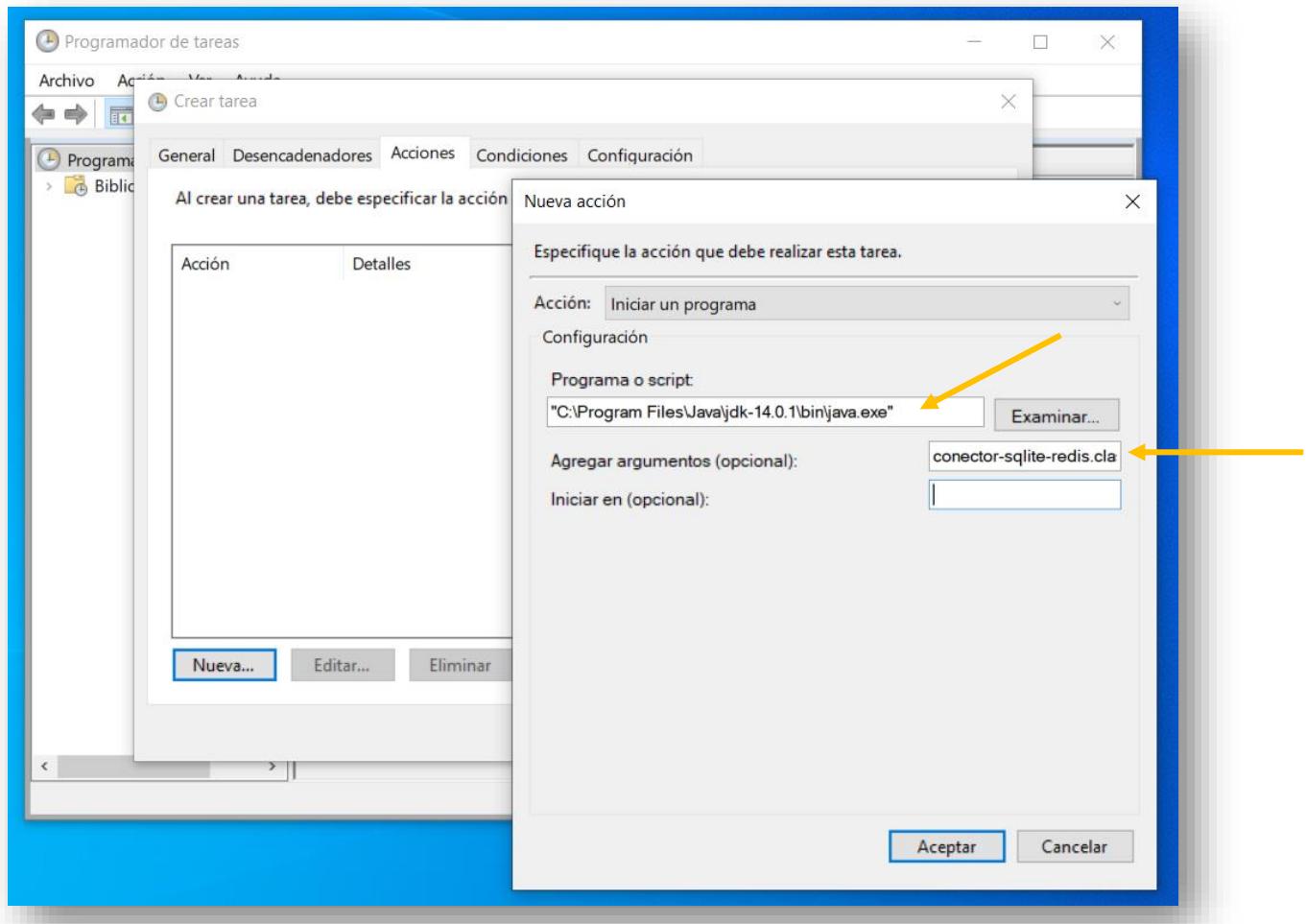
Modificare cliccando sulla scheda "Azioni" e cliccare sul pulsante "Nuovo":



Diamo i seguenti parametri:

Programma o script: connector_path

Aggiunta di argomenti: nome del connettore



I parametri di cui sopra possono variare a seconda della posizione del connettore. L'idea principale è l'esecuzione del programma **connector-sqlite-redis-v1.class** come viene normalmente eseguita sulla riga di comando:

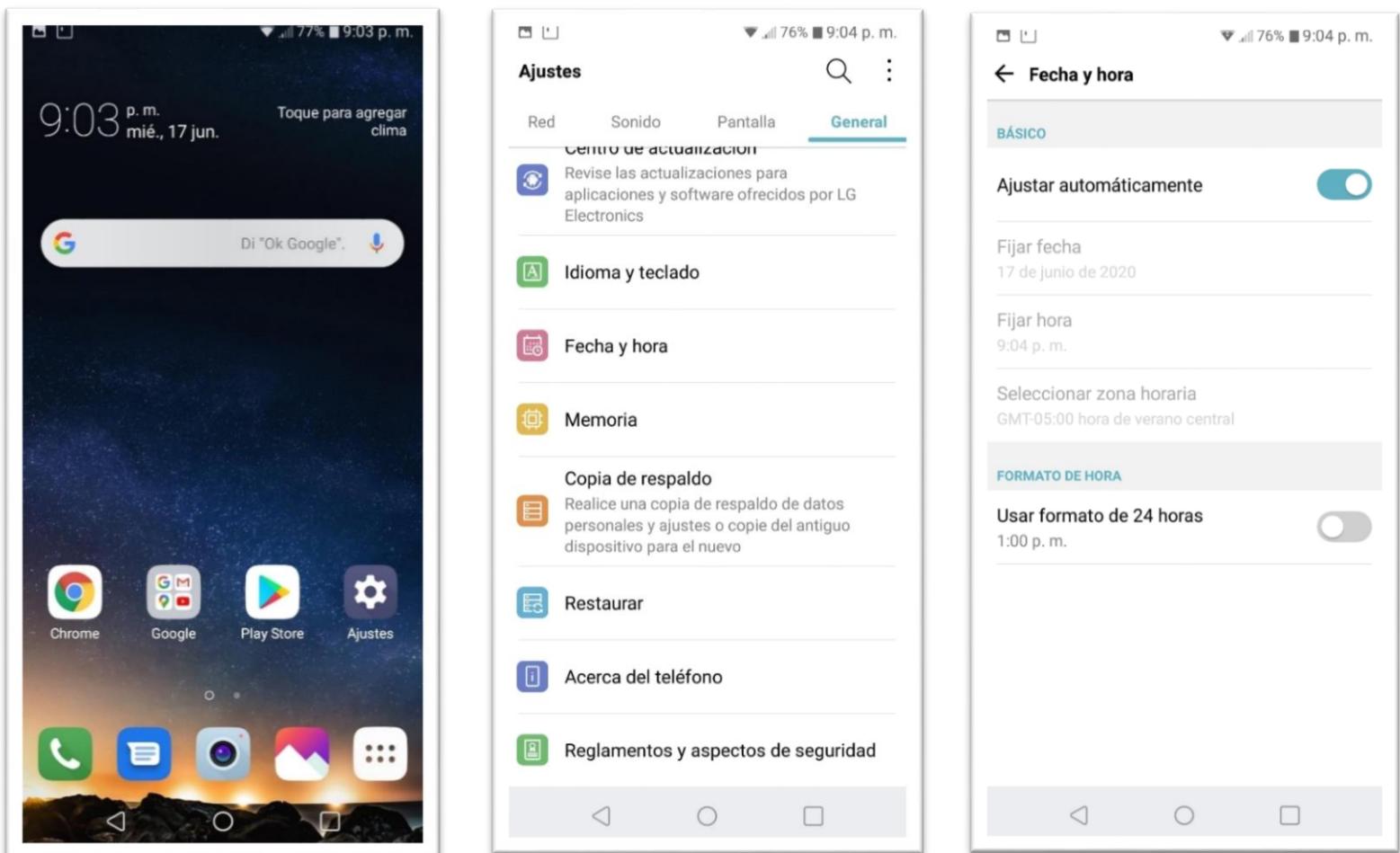
`C:\jdk_directory java connector-sqlite-redis-v1`

Per finire dobbiamo scegliere il tab "Trigger", in questo daremo i parametri di quando (giorno, ora, minuti) vogliamo che il compito venga eseguito, questi parametri si basano sulle regole di business che il sistema Mini BlocklyChain dovrà creare.

10. Sincronizzazione nei nodi del sistema (telefono cellulare) minuti e secondi.

È molto importante che tutti i nodi siano sincronizzati principalmente nella parte dei minuti e dei secondi. Dal momento che quando viene fatto l'invio o la pubblicazione in un determinato momento della coda delle transazioni tutti i nodi devono essere sincronizzati poiché questo dipenderà dal task manager che verrà stabilito nel sistema locale con lo strumento CRON da eseguire contemporaneamente in tutti i nodi, questo farà sì che tutti i nodi abbiano la stessa probabilità di poter ottenere il diritto di essere quello scelto per poter elaborare la coda delle transazioni e di poter generare il nuovo blocco per aggiungerlo alla catena di blocchi del sistema. Per sincronizzare i nodi abbiamo due opzioni.

La prima opzione della sincronizzazione del nodo, saremo in grado di farlo in modo semplice. Nel nostro dispositivo è attraverso l'opzione interna che include il sistema Android, dovremo andare nella parte di **Impostazioni > Data e ora > Regola automaticamente**



Con la configurazione precedente avremo sincronizzato in minuti e secondi tutti i nodi del sistema non importa quale paese del mondo sono già sincronizzati è basato su ogni area geografica eventualmente ciò che cambia è il tempo, ma a seconda dei minuti e secondi dovrebbe essere sincronizzato questo è sufficiente per noi per eseguire il processo di attività su base programmata con lo strumento cron in tutti i nodi per eseguire ogni certo tempo in minuti, cioè possiamo creare un compito in crontab per eseguire ogni 10 minuti o 30 minuti a seconda di ogni progetto di sistema.

Questo sarà utile quando si utilizza la rete di comunicazione "Peer to Peer", in caso di utilizzo della rete di backup questo processo non si applica in quanto la distribuzione della coda delle transazioni avviene in un modello client-server e il server è quello che controlla lo strumento di cron.

Riferimento: <https://appinventor.mit.edu/explore/blogs/karen/2016/08.html>

La seconda opzione è quella di utilizzare un'API esterna dove eseguiremo il comando Curl attraverso l'estensione (**ConnectorSSHClient**).

Il luogo in cui utilizzeremo i servizi esterni di NTP (Network Time Protocol) è il luogo in cui utilizzeremo i servizi esterni di NTP (Network Time Protocol):

<http://worldtimeapi.org/>

Ora cercheremo un modo per ottenere l'ora dai server NTP che si trovano in tutto il mondo e che ci aiuterà a far sì che tutti i nodi abbiano una query ad una certa ora nella stessa data e ora.

Esempio di interrogazione con estensione (**ConnectorSSHClient**).

\$ ricciolo "http://worldtimeapi.org/api/timezone/America/Mexico_City"

Abbiamo effettuato il collegamento al terminale Termux:



Eseguiamo il comando Curl:



Dobbiamo tener conto del fatto che il risultato del comando Curl sarà in formato JSON, qualcosa di simile al seguente risultato:

```
abbreviation: "CDT"
client_ip: "200.77.16.151"
datetime: "2020-06-18T14:16:57.750466-05:00"
day_of_week: 4
day_of_year: 170
dst: true
dst_from: "2020-04-05T08:00:00+00:00"
dst_offset: 3600
dst_until: "2020-10-25T07:00:00+00:00"
raw_offset: -21600
timezone: "America/Mexico_City"
unixtime: 1592507817
utc_datetime: "2020-06-18T19:16:57.750466+00:00"
utc_offset: "-05:00"
week_number: 25
```

Inoltre il risultato potrebbe essere in formato JSON senza i dati formattati in o JSON in forma lineare come mostrato di seguito:

```
{"abbreviazione":"CDT","client_ip":"200.77.16.151","data e ora":"2020-06-18T14:19:07.216800-05:00","giorno_di_settimana":4,"giorno_dell'anno":170,"dst":true,"dst_da": "2020-04-05T08:00:00+00:00","dst_offset":3600,"dst_until": "2020-10-25T07:00:00+00:00","raw_offset": -21600,"fuso orario": "America/Mexico_City","unixtime": 1592507947,"utc_datetime": "2020-06-18T19:19:07.216800+00:00","utc_offset": "-05:00","numero_settimana": 25}.
```

In uno dei due modi precedenti dovremo filtrare le informazioni attraverso un'estensione JSON esistente come "JSONTOOLS" o utilizzare i filtri in App Inventor nell'elaborazione del testo per ottenere solo l'ora, il giorno o i secondi a seconda delle esigenze di ogni sistema. Dopo aver elaborato il risultato, si può effettuare un confronto logico e sulla base di tale confronto possiamo eseguire un compito già programmato con il servizio "cron" che in seguito vedremo la sua configurazione in ogni nodo.

Riferimento estensione JSONTOOLS:

<https://thunkableblocks.blogspot.com/2017/07/jsontools-extension.html>

Abbiamo ora esaminato due opzioni per sincronizzare l'ora dei nodi. Continueremo a configurare il servizio "cron" su ogni nodo.

Configurazione della programmazione automatica dei compiti per l'esecuzione con il servizio **CRON** su sistemi Android (nodi di sistema).

Prima di tutto dobbiamo capire come funziona il pianificatore automatico.

Il servizio cron normalmente tutti i sistemi hanno questo pre-installato e dove tutti i compiti da eseguire automaticamente sono pianificati sono in un file chiamato crontab.

Modifichiamo il file crontab con **\$ crontab -e**, useremo l'editor **vi**, all'interno usiamo il formato:

```
# m h h dom mon dow comando utente
```

dove:

- **m** corrisponde al minuto in cui lo script verrà eseguito, il valore va da 0 a 59
- **h** l'ora esatta, il formato è di 24 ore, i valori vanno da 0 a 23, essendo 0 12:00 mezzanotte.
- **dom** si riferisce al giorno del mese, ad esempio è possibile specificare 15 se si vuole correre ogni 15
- **dow** significa il giorno della settimana, può essere numerico (da 0 a 7, dove 0 e 7 sono la domenica) o le prime 3 lettere del giorno in inglese: mon, tue, wed, thu, fri, fri, sat, sun.
- **utente** definisce l'utente che eseguirà il comando, può essere root, o un utente diverso, purché abbia i permessi per eseguire lo script.
- **il comando** si riferisce al comando o al percorso assoluto dello script da eseguire, esempio: /home/user/scripts/update.sh, se si chiama uno script deve essere eseguibile

Per chiarire alcuni esempi di compiti di cron spiegati:

```
15 10 * * * utente /home/utente/scripts/aggiornamento.sh  
Eseguirete lo script update.sh alle 10:15 ogni giorno
```

```
15 22 * * * utente /home/utente/scripts/aggiornamento.sh  
Eseguirete lo script update.sh alle 22:15 ogni giorno
```

```
00 10 * * * 0 root apt-get - e aggiornare User root  
Ogni domenica alle ore 10:00 del mattino verrà eseguito un aggiornamento.
```

```
45 10 * * * radice del sole apt-get - e aggiornamento  
L'utente root eseguirà un aggiornamento ogni domenica (sole) alle 10:45.
```

Abbiamo salvato le modifiche nell'editor e con questo abbiamo finito la configurazione del servizio cron. Nel caso in cui non abbiate il cron installato nel sistema potete farlo con il seguente comando:

\$ apt installare cron

2. Installazione e configurazione dei Nodi di rete - Telefoni cellulari.

Iniziamo con la rete di comunicazione per i nodi che utilizzeranno Mini BlocklyChain.

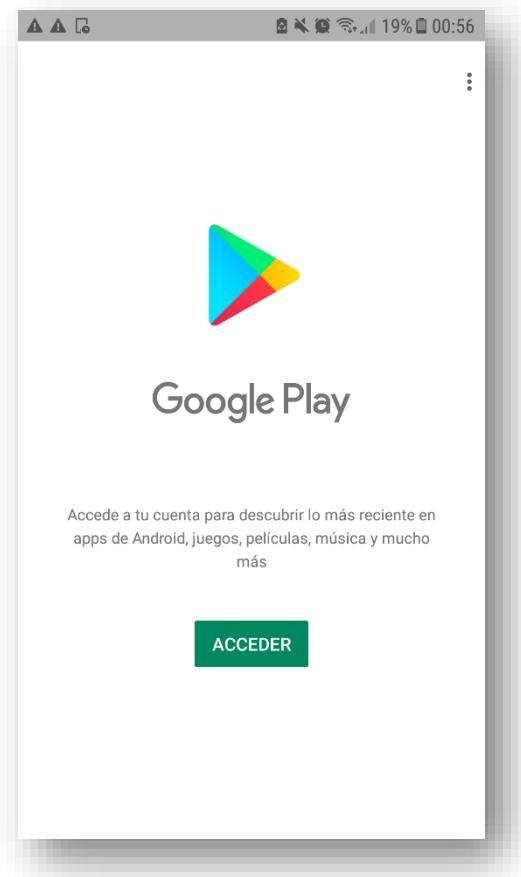
Per prima cosa abbiamo bisogno di un ambiente Linux in quanto ogni sistema Android è basato su Linux per la sicurezza e la flessibilità degli strumenti, useremo il terminale "Termux" che contiene quell'ambiente in cui installeremo la rete di comunicazione.

Termux è un emulatore Linux dove installeremo i pacchetti necessari per creare la nostra rete di comunicazione tra i nodi.

Uno dei principali vantaggi dell'utilizzo di Termux è la possibilità di installare programmi senza dover "ruotare" il telefono cellulare (Smartphone), il che assicura che nessuna garanzia del produttore vada persa a causa di questa installazione.

Installazione Termux.

Dal tuo cellulare, vai all'applicazione con l'icona di Google Play (play.google.com).



Ricerca per applicazione "Termux", selezionarla e avviare il processo di installazione.

The image consists of two side-by-side screenshots from a mobile application store, likely Google Play, displaying the details of the "Termux" app.

Screenshot 1 (Left): This screenshot shows the search results for "termux". The top bar indicates a battery level of 70% and the time is 12:28 a.m. The search bar contains the text "termux". Below the search bar, the app "Termux" by Fredrik Fornwall is listed. It has a rating of 4.4 stars based on 59k reviews and over 5 million downloads. The app icon is a black square with a white right-pointing arrow. A green "Instalar" (Install) button is visible. Below the stats are three screenshots of the app's interface, which shows a terminal window with various Linux commands and output. A descriptive text below the screenshots reads: "Emulador de terminal y entorno Linux."

Screenshot 2 (Right): This screenshot shows the detailed page for the "Termux" app. The top bar shows a battery level of 69% and the time is 12:31 a.m. The title "Termux" and "Detalles" (Details) are at the top. Below the title, there is a section titled "Más información" (More information) with a "Todos" (All) button and a "Más información" (More information) link. The main body of the page is titled "Información" (Information) and lists the following details:

Información	Datos
Versión	0.94
Actualización:	24 mar. 2020
Descargas	Más de 5,000,000 descargas
Tamaño de descarga	16.67 MB
Ofrecido por	Fredrik Fornwall
Lanzamiento:	30 jun. 2015
Permisos de la app	Ver más

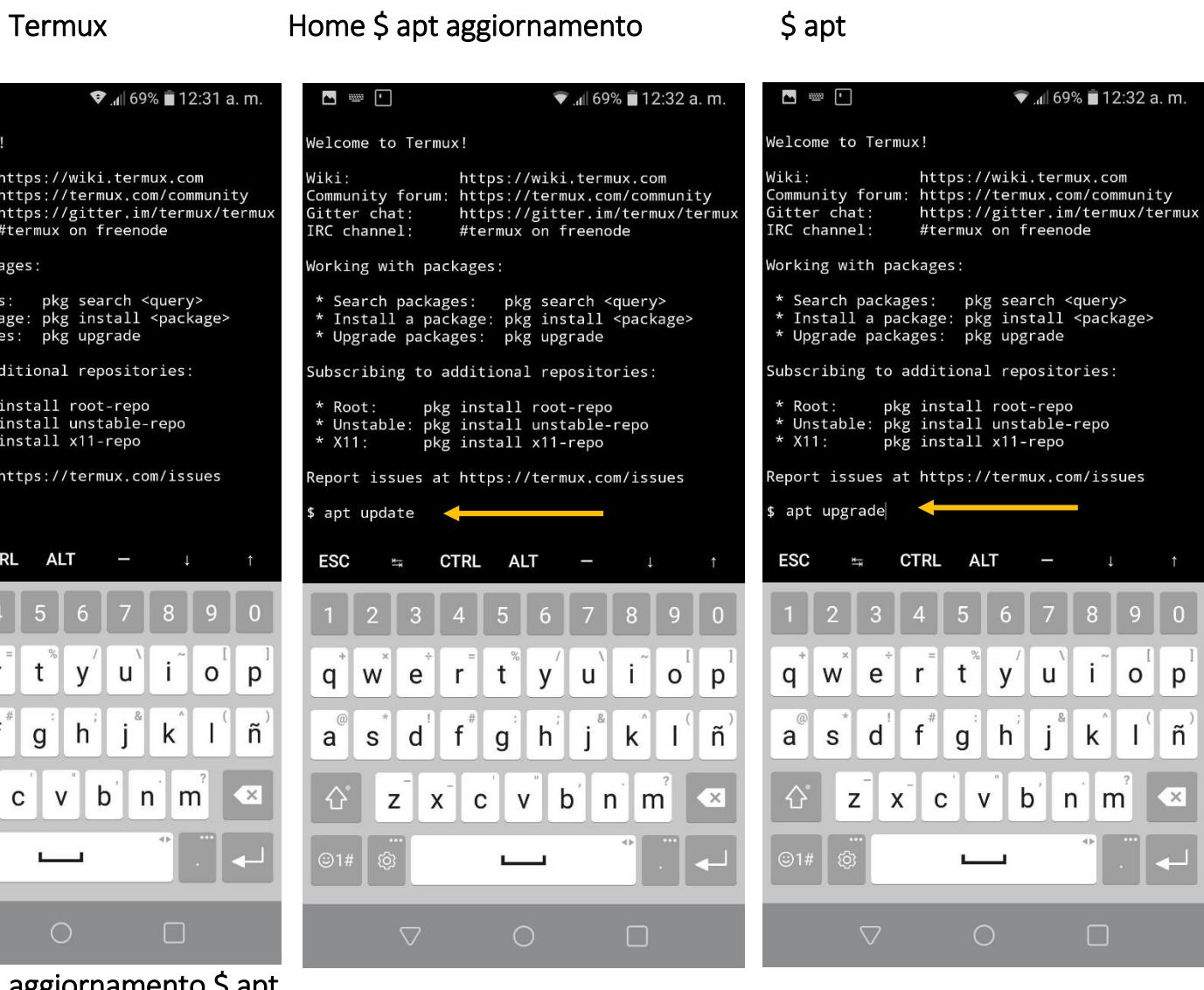
Inizio dell'applicazione Termux.

Dopo l'avvio dovremo eseguire i seguenti due comandi per eseguire gli aggiornamenti dell'emulatore del sistema operativo Linux:

\$ apt aggiornamento

\$ apt aggiornamento

Confermare tutte le opzioni Y(Sì)...



11. Configurazione della memoria all'interno di Termux.

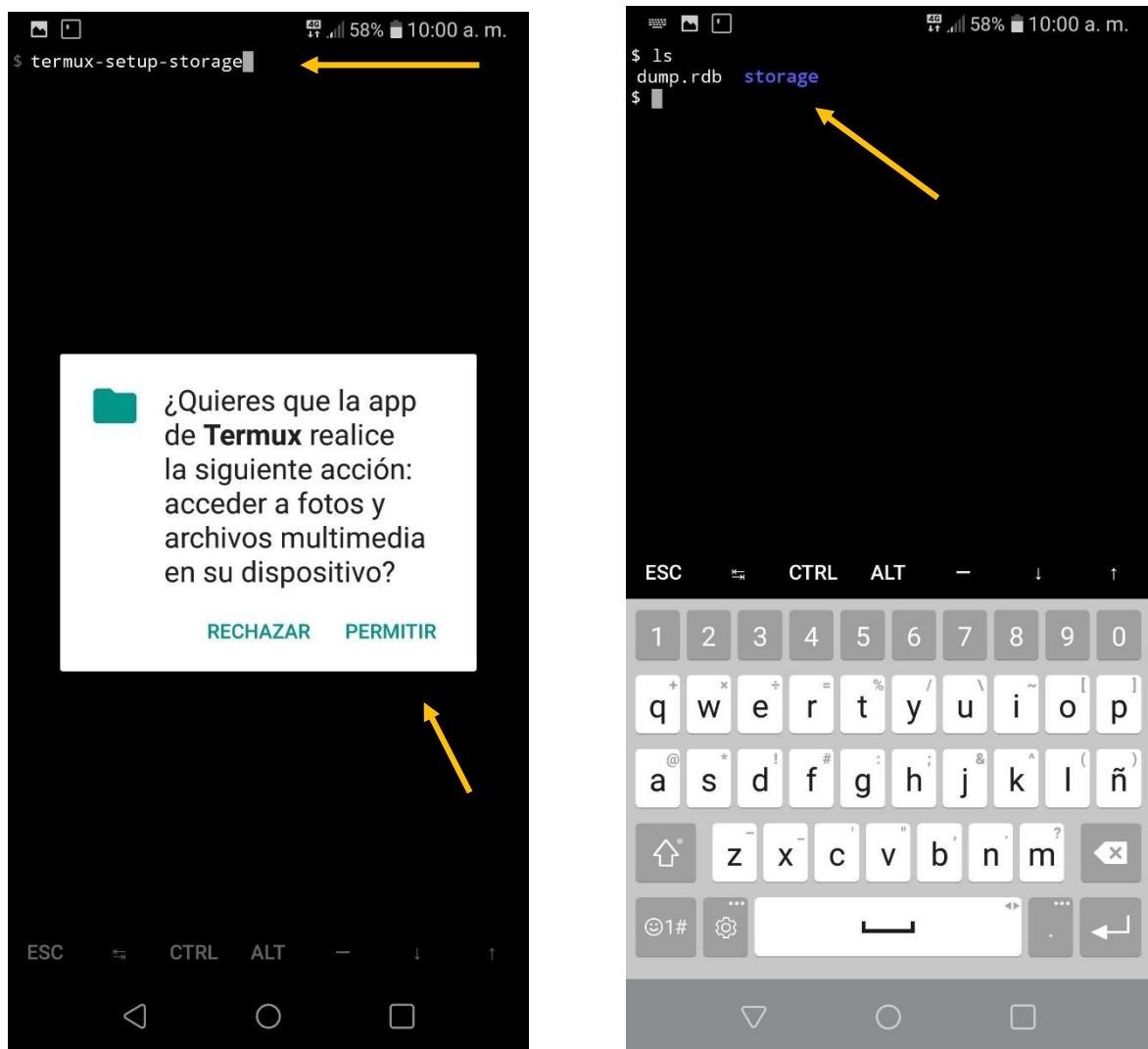
Dopo aver aggiornato e aggiornato il sistema Termux, inizieremo a configurare come visualizzare la memoria interna del telefono nel sistema Termux. Questo vi aiuterà a scambiare informazioni tra Termux e le nostre informazioni nel telefono.

Questo può essere fatto in modo semplice e veloce eseguendo il seguente comando su un terminale Termux.

`$ termux-setup-storage`

Quando si esegue il comando precedente, appare una finestra che chiede di confermare la creazione di un **archivio** virtuale (directory) in Termux. Verifichiamo dando il comando:

`$ ls`



12. Installazione di rete "Tor" e installazione "Syncthing".

```
$ apt install tor
```

```
$ apt installare la sincronizzazione
```

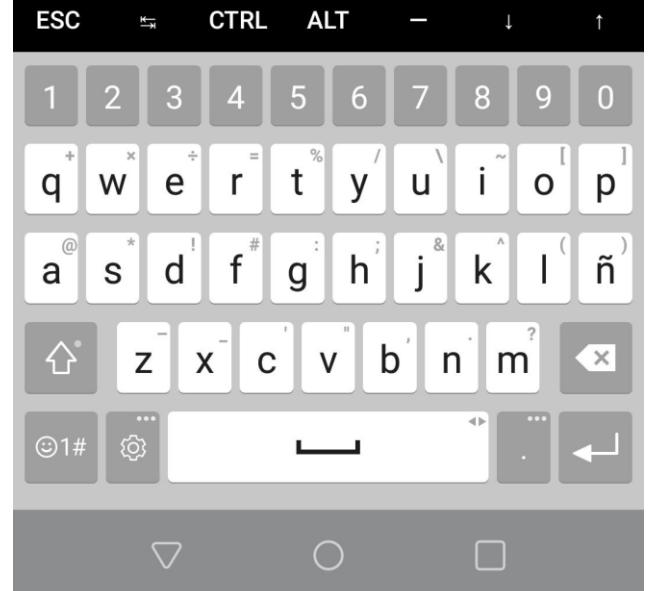
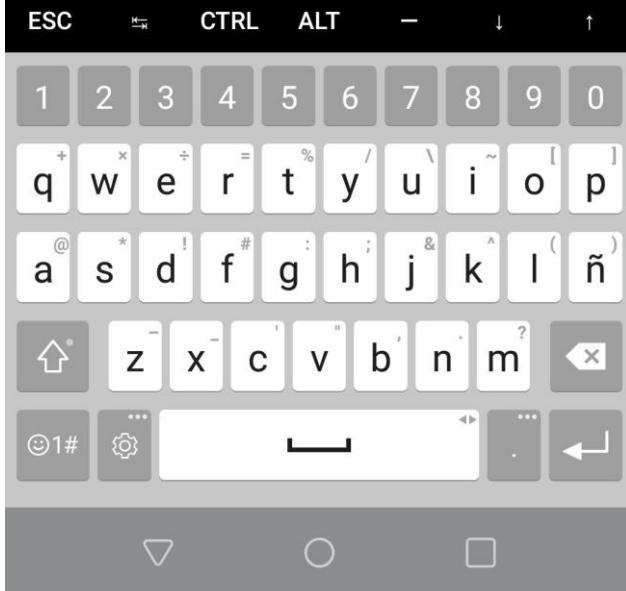
Accettare l'installazione digitando Y maiuscola in entrambi i casi se richiesto...

```
$ apt install tor
```

```
$ apt install tor
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
libevent
The following NEW packages will be installed:
libevent tor
0 upgraded, 2 newly installed, 0 to remove and 0
not upgraded.
Need to get 2319 kB of archives.
After this operation, 12.6 MB of additional disk
space will be used.
Do you want to continue? [Y/n] Y
```

```
$ apt install syncthing
```

```
$ apt install syncthing
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
syncthing
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 6407 kB of archives.
After this operation, 19.3 MB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm syncthing arm 1.5.0 [6407
kB]
27% [1 syncthing 2193 kB/6407 kB 34%]
```



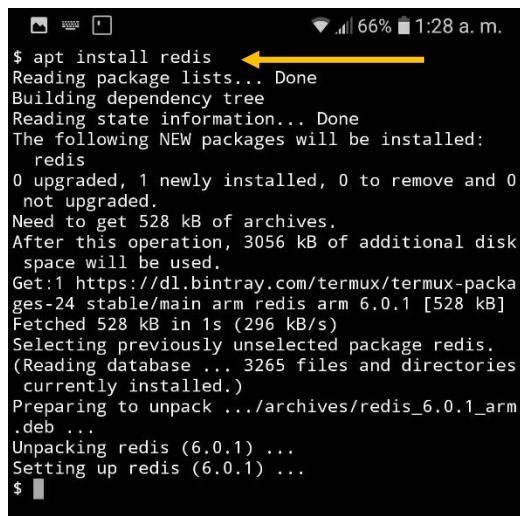
13. Installazione del database "Redis" e del server SSH (Secure Shell).

```
$ apt install redis
```

```
$ apt install openssh
```

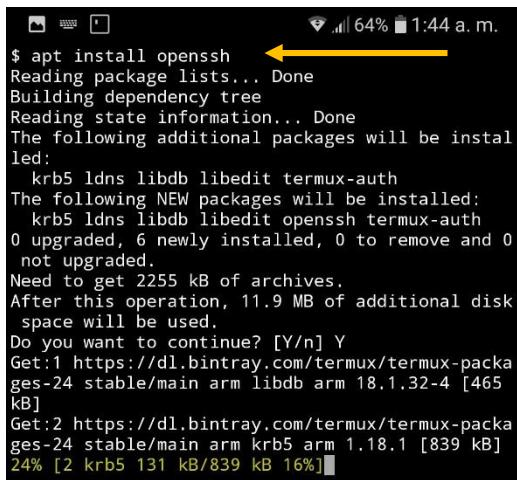
```
$ apt install sshpass
```

\$ apt install redis



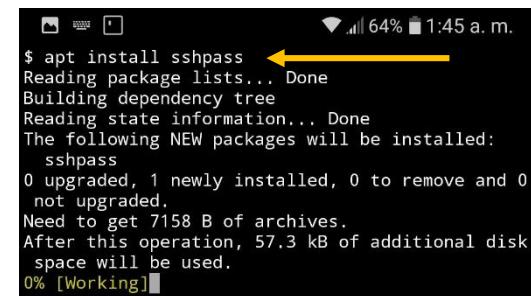
```
$ apt install redis
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  redis
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 528 kB of archives.
After this operation, 3056 kB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm redis arm 6.0.1 [528 kB]
Fetched 528 kB in 1s (296 kB/s)
Selecting previously unselected package redis.
(Reading database ... 3265 files and directories
currently installed.)
Preparing to unpack .../archives/redis_6.0.1_arm
.deb ...
Unpacking redis (6.0.1) ...
Setting up redis (6.0.1) ...
$
```

\$ apt install openssh



```
$ apt install openssh
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be instal
led:
  krb5-libs libdb5 libedit termux-auth
The following NEW packages will be installed:
  krb5-libs libdb5 libedit openssh termux-auth
0 upgraded, 6 newly installed, 0 to remove and 0
not upgraded.
Need to get 2255 kB of archives.
After this operation, 11.9 MB of additional disk
space will be used.
Do you want to continue? [Y/n] Y
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm libdb5 arm 18.1.32-4 [465
kB]
Get:2 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm krb5-libs arm 1.18.1 [839 kB]
24% [2 krb5-libs 131 kB/839 kB 16%]
```

\$ apt install sshpass



```
$ apt install sshpass
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  sshpass
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 7158 B of archives.
After this operation, 57.3 kB of additional disk
space will be used.
0% [Working]
```

Abbiamo terminato l'installazione della rete di comunicazione, proseguiamo con la configurazione dei pacchetti: Tor, Syncthing e Redis DB.

14. Configurazione del server SSH su cellulare (smartphone).

Permetteremo al server SSH del cellulare di collegarsi dal nostro PC al cellulare e di lavorare in modo più veloce e confortevole, inoltre servirà a verificare che il servizio del server SSH nel cellulare funzioni correttamente poiché lo utilizzeremo nella rete di comunicazione del Mini BlocklyChain.

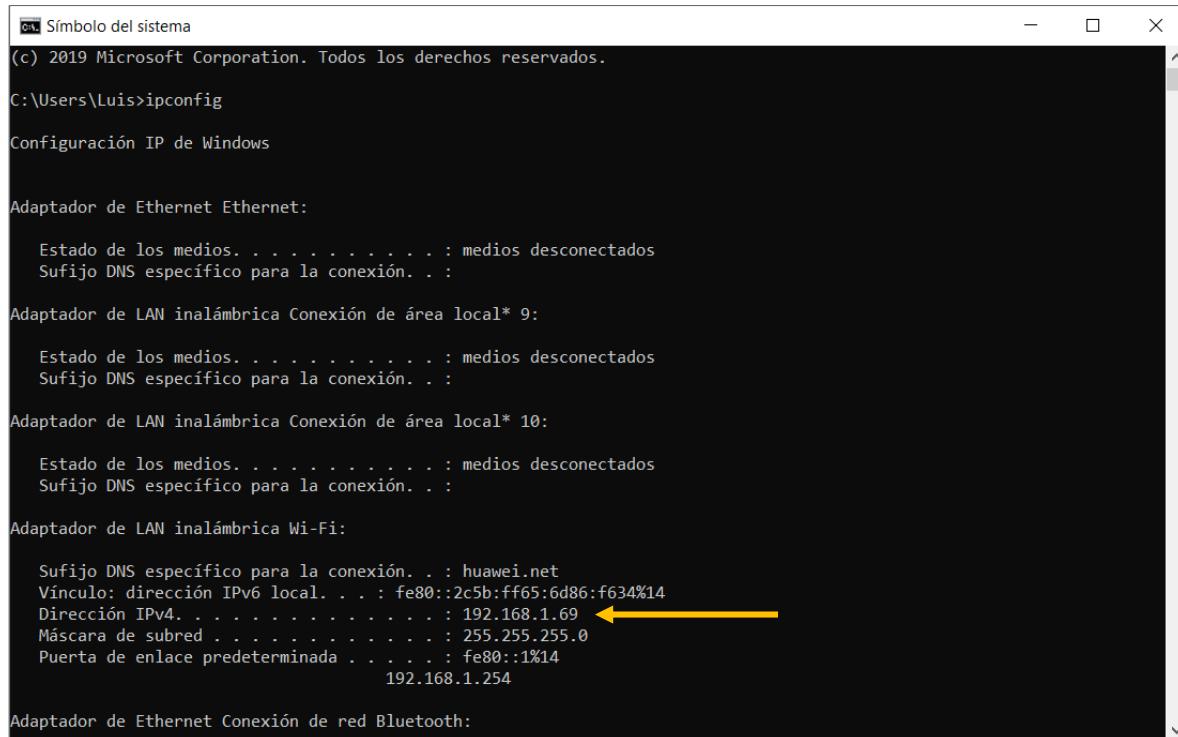
La prima cosa da fare è collegare il cellulare e il PC alla **stessa rete WiFi** in modo che possano vedersi. Gli IP o gli indirizzi devono essere simili a 192.168.XXX.XXX.XXX i valori XXX sono numeri variabili che vengono assegnati in modo casuale in ogni computer.

Questo esempio è stato testato su un cellulare LG Q6 e un PC con Windows 10 Home.

Controllare l'IP o l'indirizzo che il PC ha collegato al WiFi dobbiamo aprire un terminale in Windows.

Nel pannello inferiore dove la lente di ricerca è scritta cmd e premere il tasto Invio. Si aprirà un terminale e in esso scriveremo il comando:

C:Nome_utente> ipconfig



```
Símbolo del sistema
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Luis>ipconfig

Configuración IP de Windows

Adaptador de Ethernet Ethernet:

Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . :

Adaptador de LAN inalámbrica Conexión de área local* 9:

Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . :

Adaptador de LAN inalámbrica Conexión de área local* 10:

Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . :

Adaptador de LAN inalámbrica Wi-Fi:

Sufijo DNS específico para la conexión. . . : huawei.net
Vínculo: dirección IPv6 local. . . : fe80::2c5b:ff65:6d86:f634%14
Dirección IPv4. . . . . : 192.168.1.69
Máscara de subred . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . : fe80::1%14
192.168.1.254

Adaptador de Ethernet Conexión de red Bluetooth:
```

Ci mostrerà l'IP assegnato al PC in questo è 192.168.1.69 ma questo è molto probabilmente diverso in ogni caso. NOTA: L'indirizzo dove c'è scritto "indirizzo IPv4" deve essere preso, da non confondere con il Gateway.

Ora nel caso del telefono cellulare nel terminale Termux dobbiamo digitare il seguente comando per conoscere il nome del nostro utente che useremo per connetterci al server SSH che ha il nostro telefono, eseguiamo il seguente comando:

\$ whoami

In seguito dobbiamo dare una password a questo utente, quindi dobbiamo eseguire il seguente comando:

\$ password

Ci chiederà di digitare una password e premere Invio, di nuovo ci chiede la password, noi la confermiamo e premiamo Invio, se ha avuto **successo "La nuova password è stata impostata con successo"** in caso di marcatura di un errore è possibile che la password non sia stata digitata correttamente. Eseguire di nuovo la procedura.

E poi per sapere che IP abbiamo in Termux digitiamo il seguente comando, l'IP è dopo la parola "inet":

\$ ifconfig -a

```
$ whoami
u0_a263
$ passwd
New password:
Retype new password:
New password was successfully set.
$ 
```



```
e 0
      TX packets 0 bytes 0 (0.0 B)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST
> mtu 1500
      inet 192.168.1.68 netmask 255.255.255.0
      broadcast 192.168.1.255
      inet6 fe80::257:c1ff:fee6:3051 prefixlen 64 scopeid 0x20<link>
          unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          txqueuelen 1000 (UNSPEC)
          RX packets 908745 bytes 947916536 (904.0 MiB)
          RX errors 0 dropped 0 overruns 0 frame errors 0
      TX packets 601034 bytes 93496881 (89.1 MiB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
$ 
```

Ora è il momento di avviare il servizio di server SSH sul vostro telefono in modo da poter ricevere le sessioni dal vostro PC. Eseguiamo il seguente comando nel terminale Termux, questo comando non dà alcun risultato.

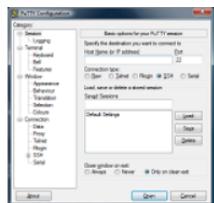
\$ sshd



Ora dovremo installare sul PC un programma che comunichi con il server SSH del telefono dal PC.

Dobbiamo andare su <https://www.putty.org>

Seleziona dove si trova il link "Puoi scaricare PuTTY qui".

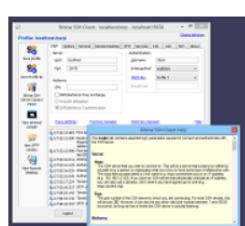


Download PuTTY

PuTTY is an SSH and telnet client, developed originally by Simon Tatham for the with source code and is developed and supported by a group of volunteers.

You can download PuTTY [here](#).

Below suggestions are independent of the authors of PuTTY. They are *not* to be seen as



Bitvise SSH Client

Bitvise SSH Client is an SSH and SFTP client for Windows. It is developed and supported prof supports all features supported by PuTTY, as well as the following:

- graphical SFTP file transfer;
- single-click Remote Desktop tunneling;
- auto-reconnecting capability;
- dynamic port forwarding through an integrated proxy;
- an FTP-to-SFTP protocol bridge.

Bitvise SSH Client is **free to use**. You can [download it here](#).

Scegliete la versione a 32 bit, non importa se il vostro sistema è a 64 bit funzionerà bene.

Download PuTTY: latest release

[Home](#) | [FAQ](#) | [Feedback](#) | [Licence](#) | [Updates](#) | [Mirrors](#)
Download: [Stable](#) · [Snapshot](#) | [Docs](#) | [Changelog](#)

This page contains download links for the latest released version of PuTTY. Currently this is 0.73, released on 2019-09-29.

When new releases come out, this page will update to contain the latest, so this is a good page to bookmark or link to. Alternative

Release versions of PuTTY are versions we think are reasonably likely to work well. However, they are often not the most up-to-date ones. See the [development snapshots](#), to see if the problem has already been fixed in those versions.

Package files

You probably want one of these. They include versions of all the PuTTY utilities.

(Not sure whether you want the 32-bit or the 64-bit version? Read the [FAQ entry](#).)

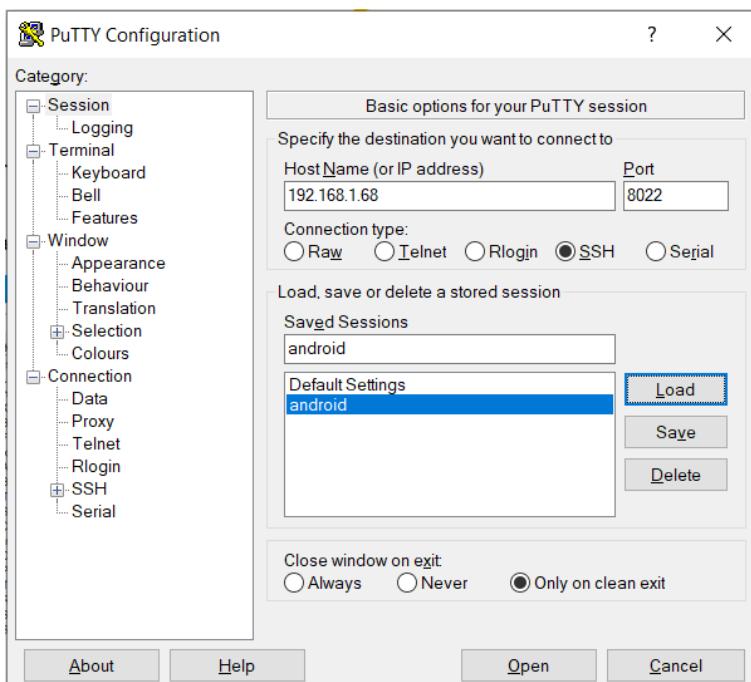
MSI ('Windows Installer')

32-bit:	putty-0.73-installer.msi	(or by FTP)	(signature)
64-bit:	putty-64bit-0.73-installer.msi	(or by FTP)	(signature)

Unix source archive

.tar.gz:	putty-0.73.tar.gz	(or by FTP)	(signature)
----------	-----------------------------------	------------------------------	-------------------------------

Una volta scaricato sul PC, eseguirlo e installarlo con le opzioni predefinite. Poi avviare l'applicazione PuTTY.



In questa sessione inseriremo i dati del nostro server Openssh che abbiamo installato nel cellulare.

Inserire l'IP del cellulare.

HostName o indirizzo IP:

192.168.1.68 (esempio IP)

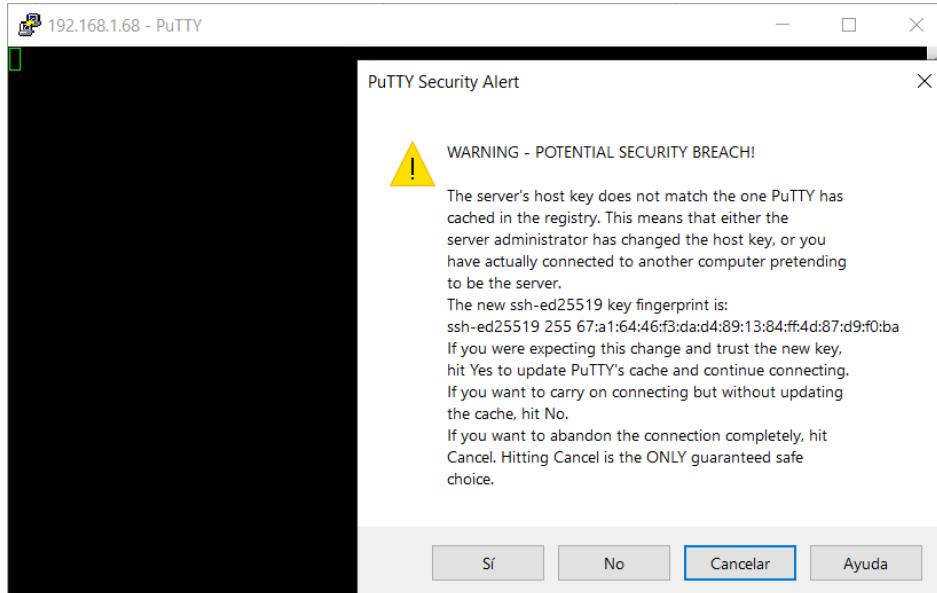
Porto:

8022 (Porta predefinita del server mobile SSH).

Possiamo dare un nome alla sessione in "Sessioni salvate" e cliccare sul pulsante Salva.

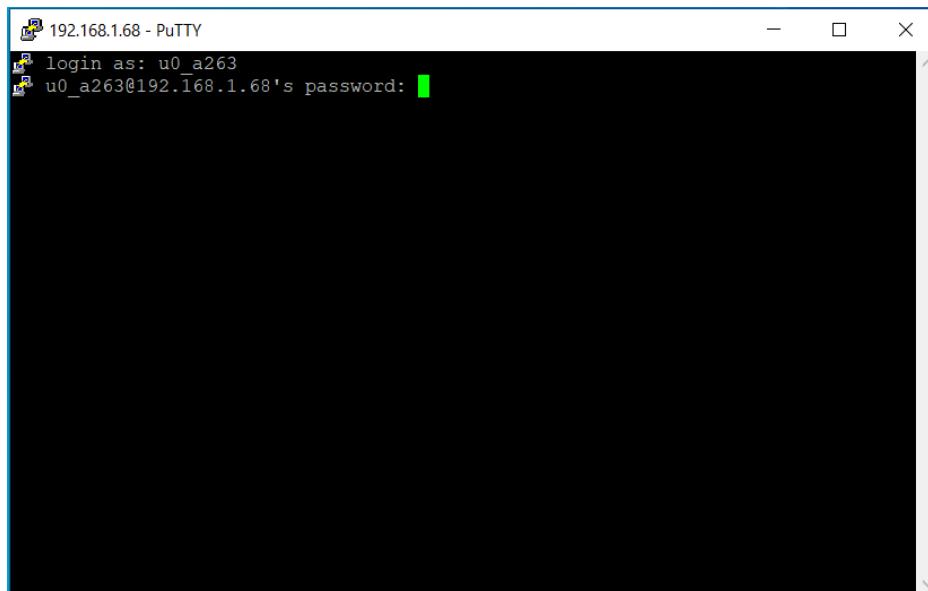
Più avanti nella parte inferiore si preme per aprire una connessione al server dando il pulsante "Apri".

Al primo collegamento sul PC vi verrà richiesta la conferma della chiave di cifratura delle informazioni cliccando sul tasto "Sì".

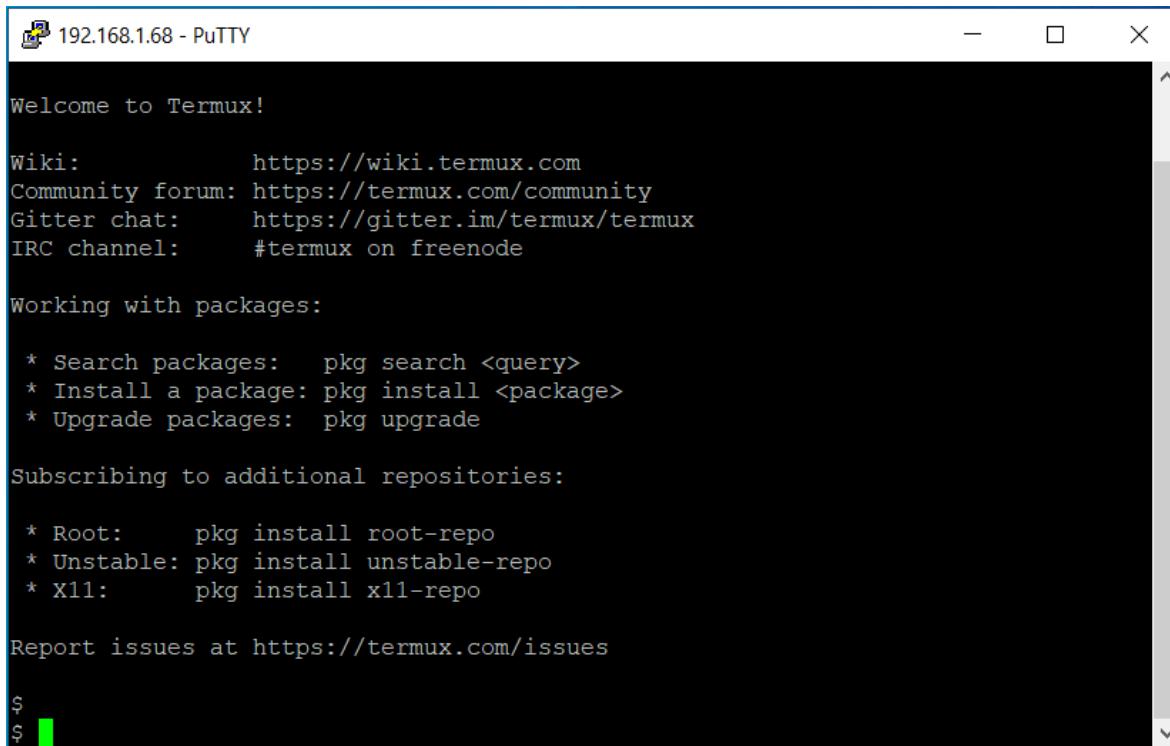


Più tardi ci verrà chiesto l'utente con cui ci collegheremo. Utilizzeremo le informazioni che abbiamo ottenuto in precedenza (utente e password).

Nel **Login come**: dobbiamo inserire il nostro utente e dare Enter, poi chiederemo di nuovo la password dando il tasto Enter.



Se i dati erano corretti, ci troveremo in una sessione SSH (Secure Shell) eseguita dal PC (Client) sul telefono (SSH Server).



```
Welcome to Termux!

Wiki: https://wiki.termux.com
Community forum: https://termux.com/community
Gitter chat: https://gitter.im/termux/termux
IRC channel: #termux on freenode

Working with packages:

* Search packages: pkg search <query>
* Install a package: pkg install <package>
* Upgrade packages: pkg upgrade

Subscribing to additional repositories:

* Root: pkg install root-repo
* Unstable: pkg install unstable-repo
* X11: pkg install x11-repo

Report issues at https://termux.com/issues

$
```

NOTA IMPORTANTE: Ricordate che l'IP (indirizzo) del PC e l'IP (indirizzo) del telefono cellulare connesso allo stesso WiFi probabilmente cambieranno ogni volta che ci disconnettiamo e ci ricolleghiamo, quindi dobbiamo controllare due volte quali indirizzi ha ogni dispositivo, questo assicurerà il successo della connessione tra i dispositivi attraverso il server SSH del telefono e il PC (Client).

Finora siamo stati in grado di connetterci solo alla stessa rete WiFi, ma se spostiamo il nostro telefono al di fuori della stessa rete del PC, non saremo in grado di connetterci perché ci sono diverse reti coinvolte nel passare attraverso altri dispositivi di comunicazione più complicati. Risolveremo questo problema quando configureremo la rete "Tor".

Ricordate che questa connessione viene effettuata solo per verificare il servizio del server che abbiamo installato sul telefono e per avere un ambiente di lavoro più confortevole con una sessione remota da PC al telefono.

15. Configurazione di rete "Tor" con servizio SSH (Secure Shell).

Con la sessione remota dal PC inizieremo a configurare la rete "Tor" con il servizio SSH abilitato.

L'importanza di avere la rete "Tor" è di dare ai dispositivi la proprietà di poter comunicare in qualsiasi parte del mondo attraverso Internet senza essere nella stessa rete WiFi, non importa dove ci troviamo saremo, saremo in grado di connetterci e formare la rete "Peer to Peer" tra i nodi che è una funzionalità essenziale dell'architettura Mini BlocklyChain.

La rete "Tor" ha molta flessibilità nella sua configurazione poiché ha diversi parametri nel suo file di configurazione "torrc" questo file è nel percorso (`$PREFIX/etc/tor/tor/torrc`) nel nostro caso con la sessione Termux dal nostro PC lo sapremo digitando il seguente comando:

```
$ echo $PREFIX
```

```
/data/dati/dati/com.termux/files/usr
```

Pertanto il file che dobbiamo modificare sarà nel percorso:

```
PREFIX/etc/tor/tor/torrc che è uguale a /data/data/com.termux/files/usr/etc/tor/tor/torrc
```

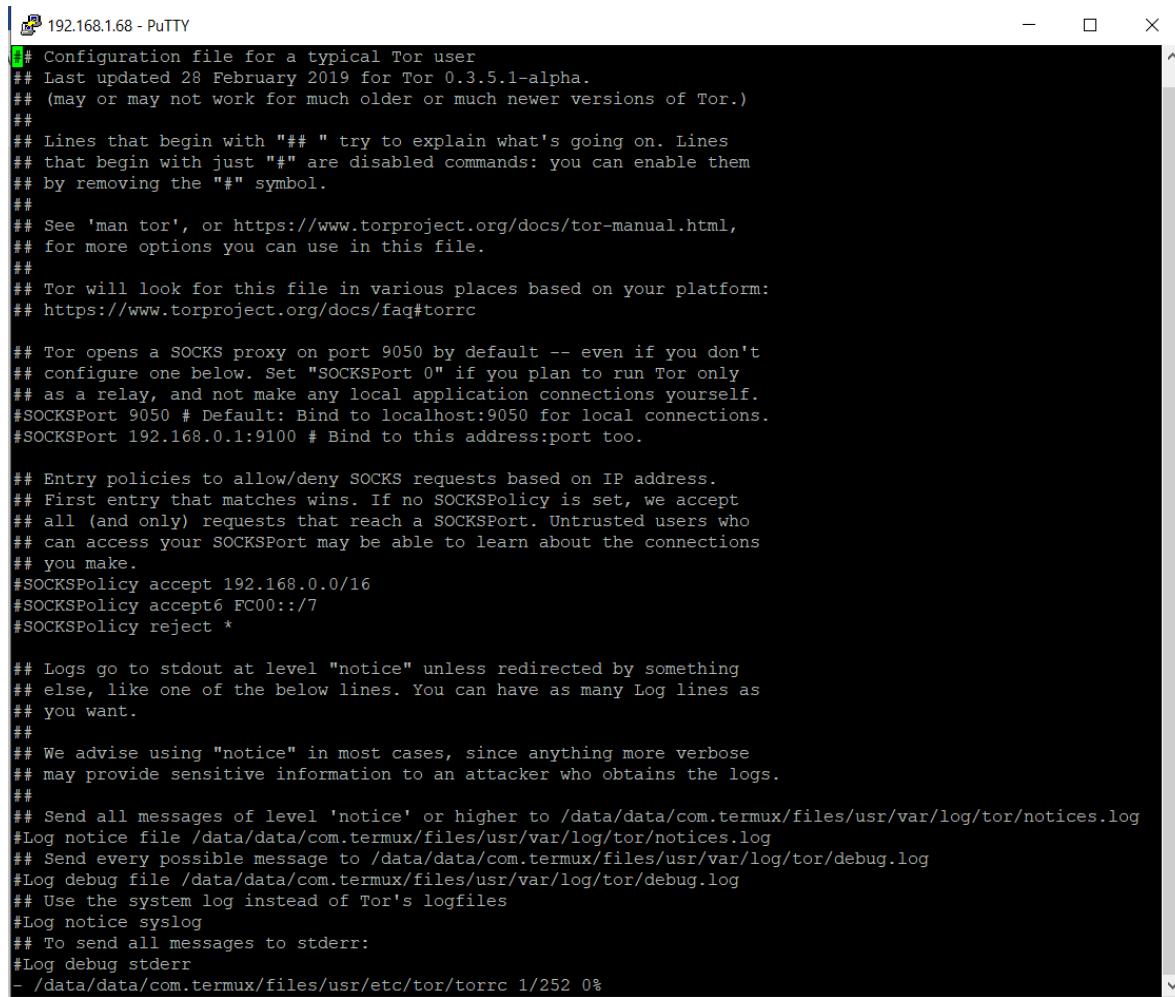
Si procede alla modifica del file di configurazione utilizzando l'editor a riga di comando "vi" eseguendo il seguente comando:

```
vi /data/dati/dati/com.termux/files/usr/etc/tor/torrc
```

Egli modificherà quel file per noi, come esempio:



File "torrc" modificato:



```
# Configuration file for a typical Tor user
## Last updated 28 February 2019 for Tor 0.3.5.1-alpha.
## (may or may not work for much older or much newer versions of Tor.)
##
## Lines that begin with "## " try to explain what's going on. Lines
## that begin with just "#" are disabled commands: you can enable them
## by removing the "#" symbol.
##
## See 'man tor', or https://www.torproject.org/docs/tor-manual.html,
## for more options you can use in this file.
##
## Tor will look for this file in various places based on your platform:
## https://www.torproject.org/docs/faq#torrc

## Tor opens a SOCKS proxy on port 9050 by default -- even if you don't
## configure one below. Set "SOCKSPort 0" if you plan to run Tor only
## as a relay, and not make any local application connections yourself.
#SOCKSPort 9050 # Default: Bind to localhost:9050 for local connections.
#SOCKSPort 192.168.0.1:9100 # Bind to this address:port too.

## Entry policies to allow/deny SOCKS requests based on IP address.
## First entry that matches wins. If no SOCKSPolicy is set, we accept
## all (and only) requests that reach a SOCKSPort. Untrusted users who
## can access your SOCKSPort may be able to learn about the connections
## you make.
#SOCKSPolicy accept 192.168.0.0/16
#SOCKSPolicy accept6 FC00::/7
#SOCKSPolicy reject *

## Logs go to stdout at level "notice" unless redirected by something
## else, like one of the below lines. You can have as many Log lines as
## you want.
##
## We advise using "notice" in most cases, since anything more verbose
## may provide sensitive information to an attacker who obtains the logs.
##
## Send all messages of level 'notice' or higher to /data/data/com.termux/files/usr/var/log/tor/notices.log
#Log notice file /data/data/com.termux/files/usr/var/log/tor/notices.log
## Send every possible message to /data/data/com.termux/files/usr/var/log/tor/debug.log
#Log debug file /data/data/com.termux/files/usr/var/log/tor/debug.log
## Use the system log instead of Tor's logfiles
#Log notice syslog
## To send all messages to stderr:
#Log debug stderr
- /data/data/com.termux/files/usr/etc/tor/torrc 1/252 0%
```

In questo file "torrc" dovremo aggiungere o usare le linee che il file ha apportando le seguenti modifiche, tre linee che sono le seguenti:

Sintassi: SOCKSPort <numero porta applicazione>

Esempio: SOCKSPort 9050

La variabile **SOCKSPort** ci dice che questo socket di comunicazione sul protocollo TCP-IP utilizzerà il dispositivo mobile (telefono) per default e la porta 9050 sarà aperta o in uso.

Sintassi: HiddenServiceDir <Directory dove verrà salvata la configurazione dell'applicazione>

Esempio: HiddenServiceDir /data/dati/dati/com.termux/files/

La variabile **HiddenServiceDir** ci dice che questa sarà la directory dove sarà memorizzata la configurazione del servizio che verrà utilizzato attraverso la rete Tor. In questa directory si trova il file di configurazione e di sicurezza del servizio, e in questa directory si trova un file chiamato **hostname**.

Possiamo vedere l'indirizzo assegnato dalla rete Tor per lo specifico servizio creato nel nostro caso è la creazione di un servizio SSH che verrà implementato sulla rete Tor, la directory che stiamo nominando come **hidden_ssh** conterrà il file **hostname**. Questa directory non ha bisogno di essere creata, perché quando avviamo il servizio di rete Tor verrà creata automaticamente.

Per vedere l'indirizzo fornito dalla rete Tor, possiamo usare il comando "more". Prima di usare questo comando, dobbiamo finire di configurare il file "torrc" e registrare il servizio di rete Tor in modo che la directory **hidden_ssh** e i file al suo interno vengano creati, come nel caso del file **hostname**.

più /data/dati/dati/com.termux/files/

Il comando di cui sopra darà come risultato un indirizzo con una stringa alfanumerica con estensione .cipolla simile a :

uqwwthff6ojdmoybyzvudoq3x2weq7k7rjitblhba3pjawk2nvjmx3wer.onion

Infine dobbiamo aggiungere la variabile **HiddenServicePort** al file di configurazione "torrc". Questo parametro indica alla rete Tor quale porta l'applicazione che stiamo firmando utilizzerà sulla rete Tor.

Sintassi: **HiddenServicePort <Porta di partenza> <Porta di partenza> <Porta locale o pubblica IP>: <Porta di partenza>**

O

HiddenServicePort <Porto di partenza>

Esempi:

HiddenServicePort 22 127.0.0.1:8022

O

HiddenServicePort 8022

Con quanto sopra abbiamo finito di configurare la rete Tor per i servizi generici e il servizio SSH (Secure Shell), che sarà utilizzato per la comunicazione di aggiornamento del database SQLite, dove salveremo i processi di validazione del nostro sistema Mini BlocklyChain.

Continuiamo con la configurazione della rete di comunicazione Mini BlocklyChain.

16. Configurazione del sistema Peer to Peer con sincronizzazione manuale.

Un'architettura "Peer to Peer" è fondamentale in un sistema tecnologico a catena di blocchi perché questa architettura fornisce due punti centrali nei processi di comunicazione del sistema, uno è quello di fornire le stesse informazioni aggiornate (sincronizzate) in qualsiasi momento in tutti i nodi, non importa se i nodi sono in una rete privata (Wifi) o in una rete pubblica (internet) o un ibrido di questi due e un secondo punto di questo tipo di architettura è che non dipendono da un intermediario (server) per trasferire, aggiornare o consultare le informazioni tra i nodi. Tutto ciò è dovuto al fatto che la comunicazione avviene direttamente tra i nodi, nel nostro caso Mini BlocklyChain la comunicazione avviene direttamente tra i telefoni che formano la rete senza un intermediario.

Per questo compito useremo lo strumento opensource Syncthing che funziona sulla base di un'architettura "Peer to Peer".

La configurazione di Syncthing per i dispositivi (telefoni cellulari) sarà vista manualmente e automaticamente. La forma manuale viene applicata in sincronizzazione con un massimo di 5 nodi, in caso di avere un gran numero di configurazioni automatiche è ottimale, il processo si concentra sulla registrazione dei nodi con i quali si vuole effettuare la sincronizzazione delle informazioni selezionate.

I requisiti per iniziare sono:

1. Hanno avviato il servizio della rete Tor.
2. (opzionale - consigliato) Avere installato un'applicazione in grado di leggere i codici QR, suggeriamo l'applicazione Android dell'inventore dell'App che è facile e semplice da installare da Google Play e più avanti in questo manuale la useremo nella sezione "Definizione e uso dei blocchi in Mini Blocklychain".
3. Di aver avviato il servizio di Sintonizzazione.



Mostriamo l'applicazione App Inventor che useremo per la lettura dei codici QR che ci serviranno in futuro per la registrazione dei nodi in Syncthing.

L'esempio seguente è stato realizzato tra due apparecchi mobili (telefoni) utilizzando i seguenti modelli:

Dispositivo mobile #1: Modello LG Q6

Dispositivo mobile #2: modello Samsung

Inizieremo con l'avviare i servizi di rete Tor e i servizi di sincronizzazione utilizzando i seguenti comandi, apriremo due terminali all'interno di Termux, ed eseguiremo ogni comando separatamente:

\$ tor

Dopo aver digitato il comando di avvio del programma di rete Tor, si dovrebbe verificare che l'esecuzione sia stata eseguita con successo, controllando che sia stata eseguita al 100%.

Eseguendo il programma Tor su un terminale Termux, verifichiamo che sia in esecuzione al 100%.

\$ tor

```

$ tor
May 23 23:00:30.932 [notice] Tor 0.4.3.5 running
on Linux with Libevent 2.1.11-stable, OpenSSL 1
.1.1g, Zlib 1.2.11, Liblzma 5.2.5, and Libzstd N
/A.
May 23 23:00:30.934 [notice] Tor can't help you
if you use it wrong! Learn how to be safe at ht
ps://www.torproject.org/download/download#warnin
g
May 23 23:00:30.939 [notice] Read configuration
file "/data/data/com.termux/files/usr/etc/tor/to
rrc".
May 23 23:00:30.970 [notice] I think we have 8 C
PUS, but only 6 of them are available. Telling T
or to only use 6. You can override this with the
NumCPUs option
May 23 23:00:30.973 [notice] Opening Socks liste
ner on 127.0.0.1:9050
May 23 23:00:30.974 [notice] Opened Socks liste
ner on 127.0.0.1:9050
May 23 23:00:30.000 [notice] Parsing GEOIP IPv4
file /data/data/com.termux/files/usr/share/tor/g
eoip.
May 23 23:00:32.000 [notice] Parsing GEOIP IPv6
file /data/data/com.termux/files/usr/share/tor/g
eoip6.
May 23 23:00:34.000 [notice] Bootstrapped 0% (st
arting): Starting
May 23 23:00:34.000 [notice] Starting with guard
context "default"
May 23 23:00:35.000 [notice] Bootstrapped 5% (co
nn): Connecting to a relay
May 23 23:00:36.000 [notice] Bootstrapped 10% (c
onn_done): Connected to a relay
May 23 23:00:36.000 [notice] Bootstrapped 14% (h
andshake): Handshaking with a relay
May 23 23:00:36.000 [notice] Bootstrapped 15% (h
andshake_done): Handshake with a relay done
May 23 23:00:36.000 [notice] Bootstrapped 20% (o
nehop_create): Establishing an encrypted directo
ry connection
May 23 23:00:36.000 [notice] Bootstrapped 25% (r

```

ESC ⌂ CTRL ALT - ↓ ↑

■ ◀ ○ □

```

ps://www.torproject.org/download/download#warnin
g
May 24 01:33:32.982 [notice] Read configuration
file "/data/data/com.termux/files/usr/etc/tor/to
rrc".
May 24 01:33:33.007 [notice] I think we have 8 C
PUS, but only 6 of them are available. Telling T
or to only use 6. You can override this with the
NumCPUs option
May 24 01:33:33.010 [notice] Opening Socks liste
ner on 127.0.0.1:9050
May 24 01:33:33.010 [notice] Opened Socks liste
ner on 127.0.0.1:9050
May 24 01:33:33.000 [notice] Parsing GEOIP IPv4
file /data/data/com.termux/files/usr/share/tor/g
eoip.
May 24 01:33:34.000 [notice] Parsing GEOIP IPv6
file /data/data/com.termux/files/usr/share/tor/g
eoip6.
May 24 01:33:35.000 [notice] Bootstrapped 0% (st
arting): Starting
May 24 01:33:37.000 [notice] Starting with guard
context "default"
May 24 01:33:38.000 [notice] Bootstrapped 5% (co
nn): Connecting to a relay
May 24 01:33:38.000 [notice] Bootstrapped 10% (c
onn_done): Connected to a relay
May 24 01:33:39.000 [notice] Bootstrapped 14% (h
andshake): Handshaking with a relay
May 24 01:33:39.000 [notice] Bootstrapped 15% (h
andshake_done): Handshake with a relay done
May 24 01:33:39.000 [notice] Bootstrapped 75% (e
nough_dirinfo): Loaded enough directory info to
build circuits
May 24 01:33:39.000 [notice] Bootstrapped 90% (a
p_handshake_done): Handshake finished with a rel
ay to build circuits
May 24 01:33:39.000 [notice] Bootstrapped 95% (c
ircuit_create): Establishing a Tor circuit
May 24 01:33:42.000 [notice] Bootstrapped 100% (d
one): Done

```

ESC ⌂ CTRL ALT - ↓ ↑

■ ◀ ○ □

Poi eseguiamo il comando di sincronizzazione:

\$ sincronizzazione

Dopo aver eseguito questo comando aprirà una pagina di amministrazione nel browser del nostro telefono se non si apre automaticamente, possiamo andare su qualsiasi browser che abbiamo installato dove navighiamo normalmente su Internet e possiamo mettere il seguente l:

<http://127.0.0.1:8384>

Aprirà la schermata di amministrazione dello strumento che ci aiuterà a sincronizzare le nostre informazioni tra tutti i nodi (telefoni) del sistema.



```
$ syncthing
[monitor] 04:02:07 INFO: Default folder created
and/or linked to new config
[start] 04:02:07 INFO: syncthing v1.5.0 "Fermium
Flea" (go1.14.2 android-arm) builder@6bdf862223
8a 2020-05-11 08:38:11 UTC
[start] 04:02:07 INFO: Generating ECDSA key and
certificate for syncthing...
[start] 04:02:08 INFO: Default folder created an
d/or linked to new config
[start] 04:02:08 INFO: Default config saved. Edi
t /data/data/com.termux/files/home/.config/synct
hing/config.xml to taste (with Syncthing stopped
) or use the GUI
[OWEPS] 04:02:08 INFO: My ID: OWEPSNA-6RZI657-SK
VOU65-V44BC5Z-CXZOJI4-C3JC7HM-IPY4V35-JQAKPAW
[OWEPS] 04:02:09 INFO: Single thread SHA256 perf
ormance is 12 MB/s using crypto/sha256 (12 MB/s
using minio/sha256-simd).
[OWEPS] 04:02:11 INFO: Hashing performance is 11
.17 MB/s
[OWEPS] 04:02:11 INFO: Migrating database to sch
ema version 1...
```



Metrica	Valor
Velocidad de descarga	0 B/s (0 B)
Velocidad de subida	0 B/s (0 B)
Estado Local (Total)	0 0 ~0 B
Oyentes	3/3
Descubrimiento	4/5
Tiempo de funcionamiento	1m
Versión	v1.5.0, android (ARM)

Otros dispositivos

- Cambios recientes
- Añadir un dispositivo

Poiché abbiamo la schermata di amministrazione "syncthing" aperta, procediamo a registrare il nodo o i nodi che vogliamo sincronizzare le informazioni. A questo punto occuperemo il programma che legge i codici QR.

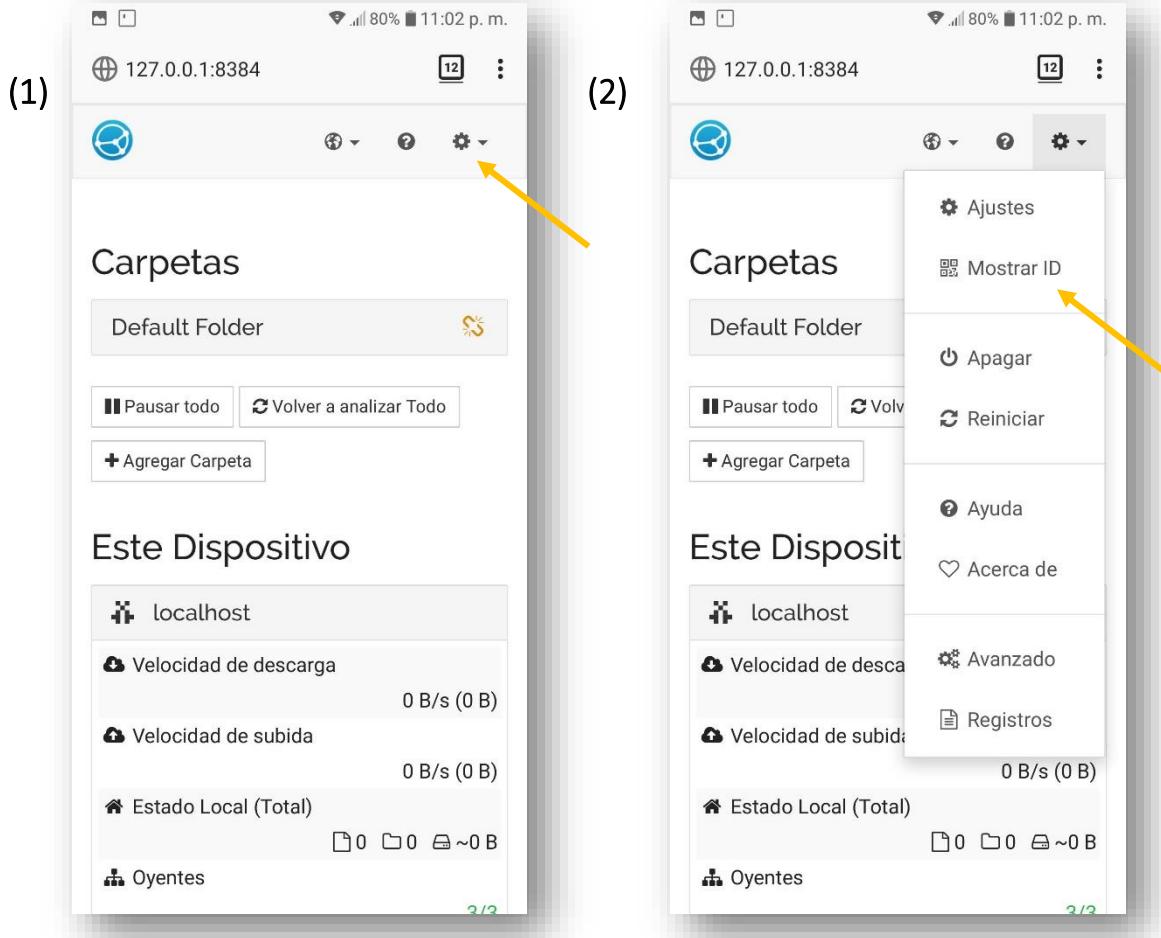
Il programma di sincronizzazione al primo avvio crea un identificatore univoco del telefono che è composto da un gruppo di otto set di caratteri alfanumerici in lettere maiuscole, questo identificatore (ID) è quello che registreremo nel nodo o nei nodi che vogliamo sincronizzare le informazioni.

Nel nostro caso l'ID del telefono LG Q6 dovrà essere registrato al telefono Sansumg e l'ID del telefono Sansumg dovrà essere registrato al LG Q6. Devono essere in entrambi i telefoni per poter funzionare correttamente.

Eseguiremo le fasi della registrazione del cellulare Sansumg sul telefono LG Q6.

Prima (1) nella parte superiore della schermata di amministrazione (browser internet) del telefono Sansumg con la sincronizzazione cliccheremo sulla scheda del menu in alto a destra.

Al secondo (2) passo vedremo un menu, in questo clicchiamo su "Mostrar ID" del Sansumg.

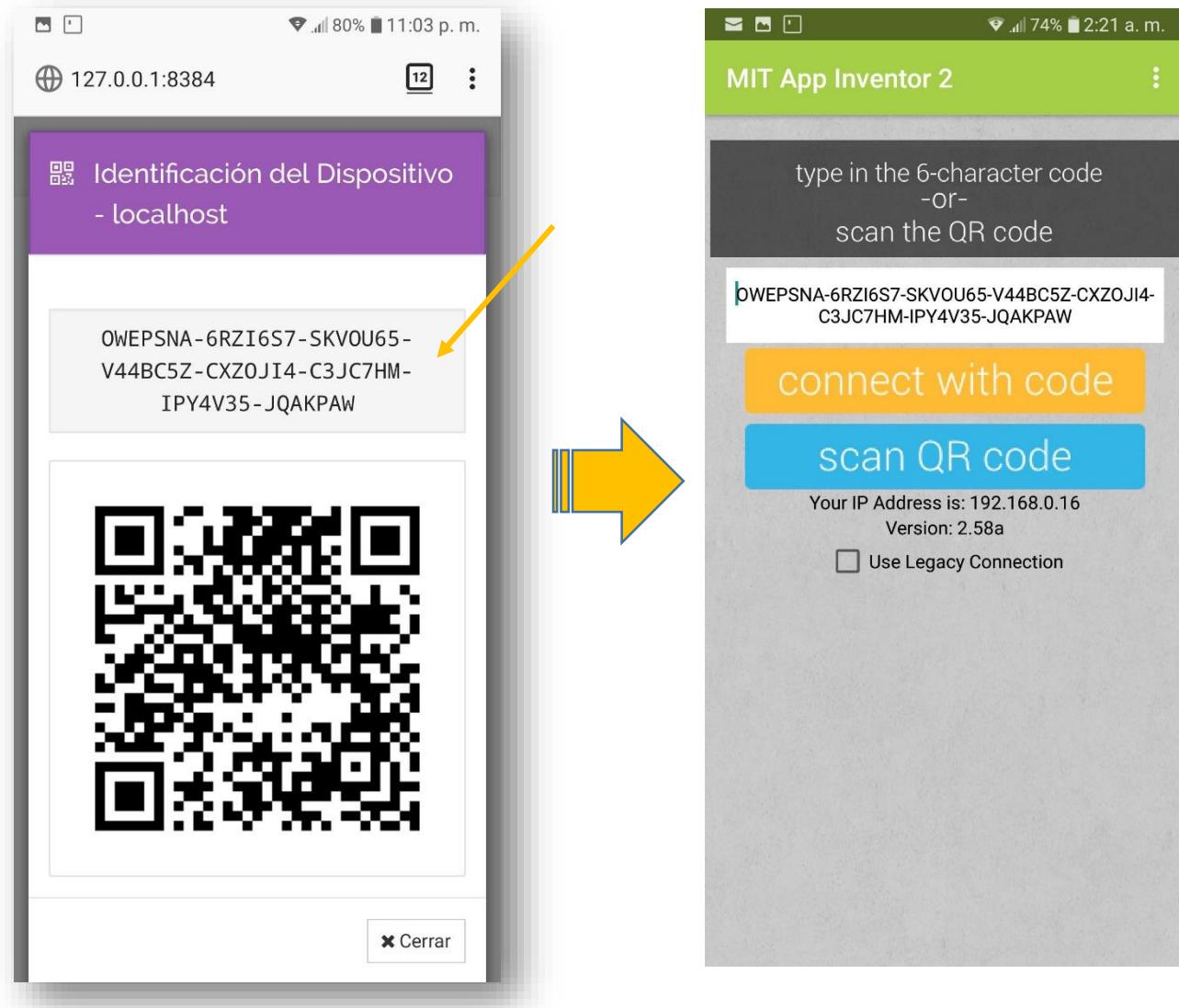


Quando si clicca su "Mostra ID" apparirà la seguente schermata che è un QR code del telefono Sansumg nel nostro caso l'ID che identifica il telefono è

OWEPSNA-6RZI6S7-SKVOU65-V44BC5Z-CXZOJI4-C3JC7HM-IPY4V35-JQAKPAW

Poi nel telefono LG Q6, il programma che ci aiuterà a catturare questo codice QR di Sansumg è quello che suggeriamo dall'applicazione App Inventor (opzionale), anche se nel caso in cui non lo avessimo possiamo anche semplicemente introdurlo manualmente quando iniziamo la registrazione nel LG Q6, tuttavia, per evitare qualsiasi errore suggeriamo di utilizzarlo.

Utilizzando l'inventore del programma App installato nel cellulare LG Q6 per catturare il codice QR dal telefono Sansumg remoto che vogliamo sincronizzare.



Poi copiamo negli appunti il codice QR nel programma di inventore di App cliccando sul codice catturato, scegliamo "SELEZIONA TUTTO" → "COPIA".

Con queste informazioni procediamo a registrare il Sansumg nella LG Q6. Nella schermata di amministrazione ci spostiamo nella parte inferiore dove c'è scritto "Altri dispositivi" e clicchiamo sul pulsante "Aggiungi un dispositivo", introduciamo l'ID del Sansumg e gli diamo un nome di registrazione.

Poi nella scheda superiore "Share" clicchiamo e in questo contrassegniamo l'opzione inferiore nella cartella "Default" con questa condivideremo una directory che è stata creata da Default chiamata Sync nel nostro dispositivo.

Poi in alto clicchiamo sulla scheda "Avanzate" e selezioniamo l'opzione di compressione dei dati sotto "Tutti i dati".

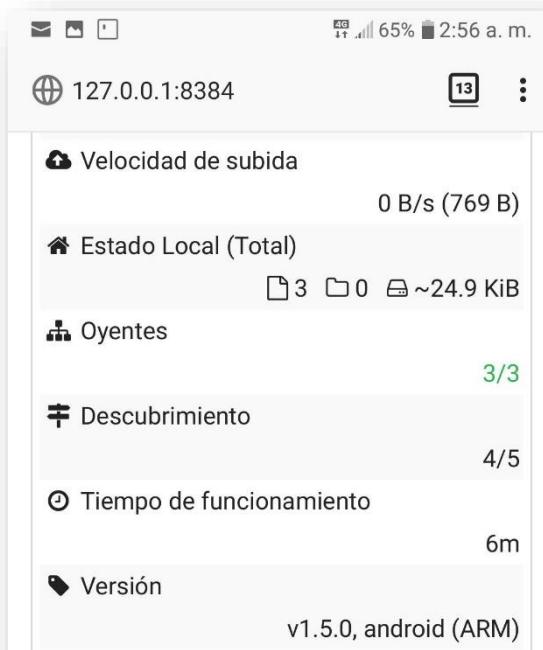
Infine clicchiamo sul pulsante di salvataggio in basso, finiamo la registrazione in un nodo, questo stesso processo deve essere fatto nel telefono o nei telefoni remoti che vogliamo sincronizzare.

The figure consists of three side-by-side screenshots of a web-based administrative interface for managing devices. The interface is in Spanish.

- Screenshot 1: Device Details**
Shows the 'ID del Dispositivo' field containing '1IQXYQ-ILHCOID-BUPFIWU-MJJUNQD' with an orange arrow pointing to it. Below it, the 'Nombre del Dispositivo' field contains 'SANSUMG' with another orange arrow pointing to it. A text note explains that the ID is found in the 'Acciones > Mostrar ID' dialog of the other device.
- Screenshot 2: Add Device**
Shows the 'Agregar el dispositivo SANSUMG' screen. It includes sections for 'General' and 'Compartiendo' settings. The 'Presentador' checkbox is checked. The 'Aceptar automáticamente' checkbox is checked. Under 'Compartir carpetas con dispositivo', 'Default Folder' is selected. At the bottom are 'Guardar', 'Mostrar QR', and 'Cerrar' buttons.
- Screenshot 3: Advanced Settings**
Shows the 'Avanzado' tab selected. It includes fields for 'Direcciones' (set to 'dynamic'), 'Compresión' (set to 'Todos los datos'), and 'Límites de velocidad del dispositivo'. The 'Guardar', 'Mostrar QR', and 'Cerrar' buttons are at the bottom.

Al momento della memorizzazione e della registrazione in entrambi i telefoni aspetteremo un periodo massimo di 30 secondi in cui i dispositivi si trovano e la connessione tra i dispositivi apparirà come buona dando una conferma in verde.

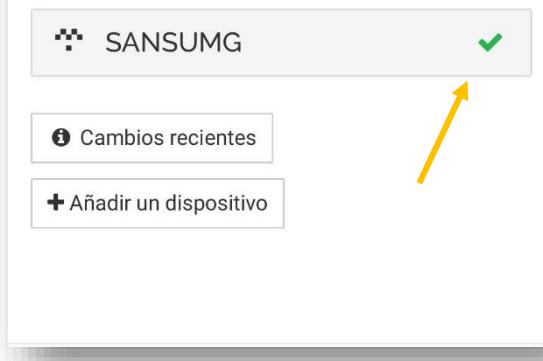
Dispositivo #1 LG Q6



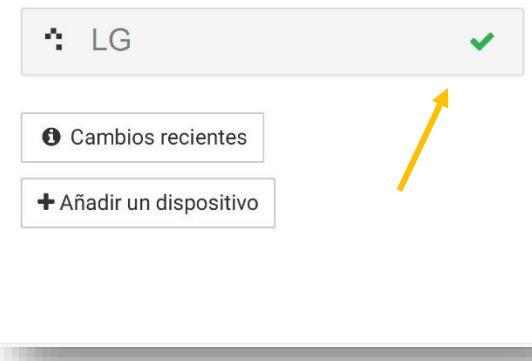
Dispositivo #2 Sansumg



Otros dispositivos



Otros dispositivos



Tutte le informazioni nella directory **Sync** che si trova nel percorso `/data/data/com.termux/file/home/Sync` saranno sincronizzate, eventuali modifiche saranno copiate e sincronizzate.

Configurazione del sistema Peer to Peer con Syncthing automaticamente per l'uso in Mini BlocklyChain.

Ora faremo la configurazione in modo automatico, in precedenza abbiamo fatto il processo manuale questo è utile se gestiamo una quantità minima di nodi, tuttavia, nel caso di avere una grande quantità di nodi sarebbe inefficiente quindi avremo lo strumento SyncthingManager per configuralo in seguito in modo automatico usando i comandi automatici online.

Configurazione del database Redis per i nodi in modalità (Slave).

Configurazione del file `/data/data/com.termux/files/usr/etc/redis.conf` del database Redis (**Slave**) per cellulari Android.

Aggiungere le seguenti modifiche o direttive al file, salvare le modifiche e avviare il server Redis.

In primo luogo, trovare e rimuovere il carattere # dalla linea **schiavoof**. Questa direttiva prende l'indirizzo IP e la porta che si utilizza per contattare in modo sicuro il server master Redis, separati da uno spazio. Per impostazione predefinita, il server Redis ascolta su 6379 sull'interfaccia locale, ma ciascuno dei metodi di sicurezza della rete cambia in qualche modo l'impostazione predefinita per gli altri.

I valori utilizzati dipenderanno dal metodo utilizzato per proteggere il traffico di rete:

Rete isolata: utilizzare l'indirizzo IP della rete isolata e la porta Redis (6379) del server master (ad es. slave del semplice indirizzo IP_indirizzo 6379).

stordimento o spiped: utilizzare l'interfaccia locale (127.0.0.1) e la porta configurata per trainare il traffico

PeerVPN: Utilizzare l'indirizzo IP VPN del server master e la normale porta Redis.

Il generale lo cambierebbe:

slaveof ip_contact_server master_contact_port

Esempio: **slave** di 192.168.1.69 6379

masterauth your_network_master_password

Esempio: **masterauth sdfssdfsdsd12WqE34Rfgthtdfd**

requirepass your_network_slave_password

Esempio: **requirepass asdsjdsh34sds67sdFGbbnh**

Salvare ed eseguire il server dal terminale Termux con il seguente comando.

\$ redis-server redis.conf

Dopo l'esecuzione possiamo vedere come si sincronizza con il server Windows 10 (**Master**).

File di configurazione redis.conf **\$ redis-server redis.conf**

The image shows two Termux sessions side-by-side. The left session shows the command `$ redis-server redis.conf` being run, and the right session shows the Redis log output.

```

$ pwd
/data/data/com.termux/files/usr/etc
$ ls
alternatives      inputrc    redis.conf
apt              krb5.conf   ssh
bash.bashrc       motd       tls
bash_completion.d profile    tmux.conf
dump.rdb          profile.d wgetrc
$ 

```

The Redis log on the right shows the synchronization process:

```

32672:S 31 May 2020 23:50:24.130 # Server initialized
32672:S 31 May 2020 23:50:24.131 * Loading RDB produced by version 6.0.1
32672:S 31 May 2020 23:50:24.131 * RDB age 27 seconds
32672:S 31 May 2020 23:50:24.131 * RDB memory usage when created 0.39 Mb
32672:S 31 May 2020 23:50:24.132 * DB loaded from disk: 0.001 seconds
32672:S 31 May 2020 23:50:24.132 * Ready to accept connections
32672:S 31 May 2020 23:50:24.132 * Connecting to MASTER 192.168.1.69:6379
32672:S 31 May 2020 23:50:24.136 * MASTER <-> REPLICATION sync started
32672:S 31 May 2020 23:50:24.159 * Non blocking connect for SYNC fired the event.
32672:S 31 May 2020 23:50:24.166 * Master replied to PING, replication can continue...
32672:S 31 May 2020 23:50:24.236 * Partial resynchronization not possible (no cached master)
32672:S 31 May 2020 23:50:24.266 * Full resync from master: 8ea52fe3c02ae241292f0dcbb823b8febcb784:0
32672:S 31 May 2020 23:50:24.349 * MASTER <-> REPLICATION sync: receiving 578 bytes from master to disk
32672:S 31 May 2020 23:50:24.353 * MASTER <-> REPLICATION sync: Flushing old data
32672:S 31 May 2020 23:50:24.353 * MASTER <-> REPLICATION sync: Loading DB in memory
32672:S 31 May 2020 23:50:24.354 * Loading RDB produced by version 5.0.7
32672:S 31 May 2020 23:50:24.354 * RDB age 0 seconds
32672:S 31 May 2020 23:50:24.354 * RDB memory usage when created 1.84 Mb
32672:S 31 May 2020 23:50:24.355 * MASTER <-> REPLICATION sync: Finished with success

```

Installazione del tool di gestione della sincronizzazione dei nodi. Procediamo all'installazione di **SyncthingManager**.

Per prima cosa installiamo ciò di cui avrete bisogno per il corretto funzionamento di SyncthingManager.

\$ apt installare Python

\$ pip3 install -upgrade pip

\$ npm installare syncthingmanager

Lo strumento SyncthingManager ci aiuterà a sincronizzare "peer to peer" in modo automatico e non manualmente per i nodi nuovi e quelli esistenti.

```
$ apt install python
Reading package lists... Done
Building dependency tree
Reading state information... Done
python is already the newest version (3.8.3).
0 upgraded, 0 newly installed, 0 to remove and 0
not upgraded.
$ 

$ pip3 install --upgrade pip
Requirement already up-to-date: pip in /data/dat
a/com.termux/files/usr/lib/python3.8/site-pac
kes (20.1.1)
$ 

$ pip3 install syncthingmanager
Requirement already satisfied: syncthingmanager in /data/dat
a/com.termux/files/usr/lib/python3.8/site-p
ackages (0.1.0)
Requirement already satisfied: syncthing in /dat
a/d
a/com.termux/files/usr/lib/python3.8/site-p
ackages (from syncthingmanager) (2.3.1)
Requirement already satisfied: python-dateutil==2.6.1 in /data/d
a/com.termux/files/usr/lib/pyt
hon3.8/site-packages (from syncthing->syncthingm
anager) (2.6.1)
Requirement already satisfied: requests==2.18.4 in /data/d
a/com.termux/files/usr/lib/python3.8/site-p
ackages (from syncthing->syncthingmanager)
(2.18.4)
Requirement already satisfied: six>=1.5 in /data/d
a/com.termux/files/usr/lib/python3.8/site-pa
ckages (from python-dateutil==2.6.1->syncthing->
syncthingmanager) (1.15.0)
Requirement already satisfied: chardet<3.1.0,>=3
.0.2 in /data/d
a/com.termux/files/usr/lib/pyt
hon3.8/site-packages (from requests==2.18.4->sync
thing->syncthingmanager) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in
 /data/d
a/com.termux/files/usr/lib/python3.8/si
te-packages (from requests==2.18.4->syncthing->
syncthingmanager) (2.6)
Requirement already satisfied: certifi>=2017.4.1
 7 in /data/d
a/com.termux/files/usr/lib/python3
.8/site-packages (from requests==2.18.4->syncthi
ng->syncthingmanager) (2020.4.5.1)
Requirement already satisfied: urllib3<1.23,>=1.
21.1 in /data/d
a/com.termux/files/usr/lib/pyt
hon3.8/site-packages (from requests==2.18.4->sync
thing->syncthingmanager) (1.22)
$ 
```

17. Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).

App Inventor è un ambiente di sviluppo software creato da Google Labs per costruire applicazioni per il sistema operativo Android. L'utente può, visivamente e da un insieme di strumenti di base, collegare una serie di blocchi per creare l'applicazione. Il sistema è gratuito e può essere facilmente scaricato dal web. Le applicazioni create con App Inventor sono molto facili da creare perché non è richiesta la conoscenza di alcun linguaggio di programmazione.

Tutti gli ambienti attuali che utilizzano la tecnologia di Blockly, come AppyBuilder e Thunkable, tra gli altri, hanno la loro versione gratuita, il loro modo di utilizzo può avvenire tramite internet nei loro diversi siti o può essere installato anche a casa.

I blocchi che compongono l'architettura Mini BloclyChain sono stati testati in App inventor e AppyBuilder ma a causa della loro ottimizzazione del codice dovrebbero funzionare sulle altre piattaforme.

Versioni online:

App Inventor.

<https://appinventor.mit.edu/>

AppyBuilder.

<http://appybuilder.com/>

Si può pensare.

<https://thunkable.com/>

Versione da installare sul vostro computer (PC):

<https://sites.google.com/site/aprendeappinventor/instala-app-inventor>

Ambiente per gli sviluppatori dei blocchi Blockly.

<https://editor.appybuilder.com/login.php>

18. Che cos'è la Prova del Quantum (PQu)?

PoQu. - "Proof of Quantum" è un algoritmo di consenso sviluppato per Mini BlocklyChain, questo test è una variante del Test of Work (PoW) che funziona come segue.

Il Test of Quantum (PoQu) all'avvio viene eseguito con lo stesso algoritmo del "Test of Work" (PoW) si basa sul mettere in funzione il processore del dispositivo (PC, Server, Tablet o Cellulare) per ottenere una stringa di caratteri che è un puzzle matematico chiamato "hash".

Ricordate che un "hash" è un algoritmo o processo matematico che quando si introduce una frase o un qualche tipo di informazione digitale come file di testo, programma, immagine, video, suono o altri diversi tipi di informazione digitale ci dà come risultato un carattere alfanumerico che rappresenta la firma digitale che la rappresenta in modo unico e non ripetibile dei dati, l'algoritmo di hash è unidirezionale, questo significa che quando si inserisce un dato per ottenere la sua firma "hash" il suo processo inverso non può essere eseguito, avendo una firma "hash" non possiamo sapere quali informazioni è stato ottenuto questa proprietà ci dà un vantaggio di sicurezza per elaborare le informazioni che inviamo su Internet. Come funziona? Immaginate di inviare qualsiasi tipo di informazione attraverso canali non sicuri e di accompagnarla con il suo rispettivo "source hash", il ricevitore quando riceve l'informazione può ottenere l'"hash" dell'informazione ricevuta lo chiameremo "destination hash" e verificheremo con l'"hash della fonte" se entrambi gli "hash" sono gli stessi possiamo confermare che l'informazione non è stata alterata nel canale che è stato inviato, è solo un esempio in cui questo tipo di processo di sicurezza dell'informazione è attualmente utilizzato.

Attualmente esistono diversi tipi di algoritmi o processi di hash che si differenziano per il livello di sicurezza. I più usati o conosciuti sono: MD5, SHA256 e SHA512.

Esempio di SHA256:

Abbiamo una catena o una frase come segue: "Mini BlocklyChain è modulare.

Se applichiamo un hash SHA256 alla stringa precedente ci darà l'hash successivo.

f41af7e61c3b02fdd5e5c612302b62a2dd52fc38f9f9of97cb2af827e8804db8

La stringa alfanumerica di cui sopra è la firma che rappresenta la frase nell'esempio precedente

Per ulteriori esempi possiamo utilizzare il sito su internet:

<https://emn178.github.io/online-tools/sha256.html>

Nel caso dell'algoritmo "Test Work" (PoW), funziona utilizzando la potenza di calcolo per ottenere un hash predefinito.

Immaginiamo di avere il precedente "hashish" che abbiamo preso dalla catena "Mini BlocklyChain è modulare".

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9f9of97cb2afd827e8804db8

A questo "hash" all'inizio mettiamo il parametro di difficoltà che è semplicemente mettere degli zeri "0" all'inizio, cioè se diciamo che la difficoltà è di 4 avrà "**0000**" + "hash" a questo lo chiameremo "seed hash".

0000 f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2afd827e8804db8

Ora tenendo conto che conosciamo l'informazione di input che è la stringa: "Mini BlocklyChain è modulare" aggiungiamo alla fine della stringa un numero che parte da zero "0" e togliamo il suo hash a questo lo chiameremo "hash nonce":

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2afd827e8804db80

Abbiamo l'hashish nonce:

7529f3ad273fc8a9eff12183f8d6f886821900750bb6b59c1504924dfd85a7c8

Poi effettuiamo un confronto del nuovo "hash nonce" con il "seme di hash" se sono uguali il nodo che per primo trova l'uguaglianza vincerà l'esecuzione dell'elaborazione della transazione corrente. Come possiamo vedere questo processo si basa sulla probabilità e sulla forza di calcolo del dispositivo che dà al test "Proof of Work" un'equità di consenso per tutti i nodi.

Se l'"hashish di seme" non coincide con l'"hash nonce", la difficoltà viene aumentata di uno e l'"hash nonce" viene eliminato di nuovo, il numero che viene aumentato viene chiamato numero "nonce", viene confrontato con l'"hashish di seme" finché non coincidono o sono uguali.

Come possiamo vedere il numero "nonce" o aumento è quello che aiuterà ad ottenere l'"hash" dell'uguaglianza.

Basato sull'algoritmo "Test of Work" (PoW), l'algoritmo Test of Quantum (PoQu) si basa sull'ottenimento del numero "nonce" come fa PoW e utilizzando un livello minimo di difficoltà che va da 1 a 5, questo serve solo al dispositivo mobile per ottenere il diritto di essere un candidato a conquistare il consenso.

Il Quantum Test (PoQu), si attiva quando il telefono cellulare ha terminato il PoW minimo e vince il passaggio per ottenere un numero di probabilità nel sistema QRNG.

Il QRNG (Quantum Random Number Generator) è un Generatore di Numeri Casuali Quantici, questo sistema si basa sulla generazione di veri numeri casuali basati sulla meccanica quantistica è il sistema più sicuro oggi per generare tali numeri. Per maggiori dettagli si veda l'allegato "Calcolo quantistico con OpenQbit".

Mini BlocklyChain può implementare sia i tipi di concessione PoW che PoQu minimi.

Il test PoQu si basa sull'ottenimento del numero "nonce" questo numero nel test PoQu è conosciuto come "Magic Number" con questo il sistema "Peer to Peer" confermerà se il numero è corretto e poi un numero casuale sarà ottenuto con il pool di server QRNG. Questo numero casuale verrà registrato in tutti i nodi, verrà creata una lista contenente **((Node Sum /2)) +1** e da questa lista verrà scelto quello con la più alta percentuale di probabilità di essere il candidato vincitore del consenso (PoQu) e questo eseguirà la coda di transazione corrente.

L'algoritmo PoQu utilizza anche i test del **NIST** (National Institute of Standards and Technology) per assicurarsi che i numeri casuali nel QRNG siano veramente casuali.

<https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf>

In Mini BlocklyChain abbiamo implementato un blocco per PoW e un blocco per PoQu.

Questi blocchi usano un tipo di hashish: SHA256 per uso gratuito, per uso commerciale si ha un SHA512 e altri tipi di hashish come richiesto.

Per maggiori dettagli sul concetto di HASH vedi:

https://es.wikipedia.org/wiki/Funcion_hash

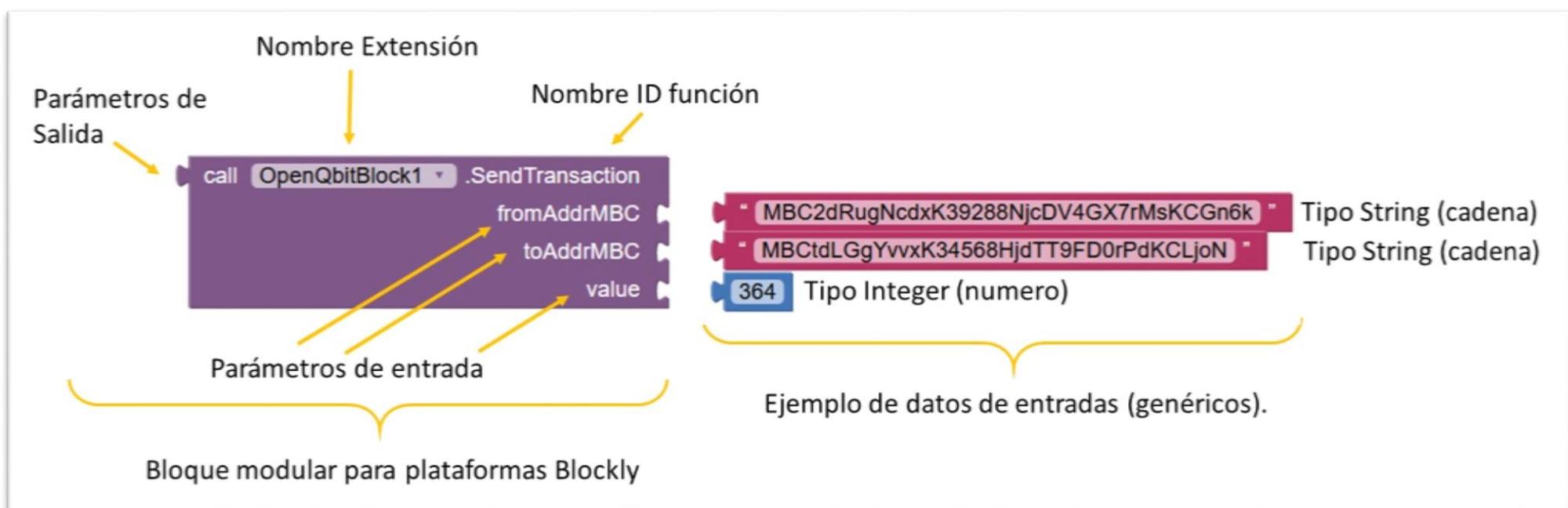
NOTA: Il Test of Work (PoW) utilizzato nei telefoni cellulari può utilizzare solo un massimo di 5 poiché l'elaborazione matematica di questi dispositivi non è dedicata come server o PC. Utilizziamo l'algoritmo PoW solo per ottenere la possibilità di ottenere il pass o il permesso di entrare nel sistema Quantum Random Number Generator (QRNG) e con esso eseguire l'algoritmo Quantum Random Number Generator (PoQu).

Sui telefoni cellulari non utilizzare una difficoltà massima di 5 in quanto il sistema potrebbe bloccarsi e non rispondere correttamente.

19. Definizione e uso dei blocchi in Mini BlocklyChain

Inizieremo spiegando la distribuzione dei dati che tutti i blocchi avranno, la loro sintassi d'uso e la loro configurazione.

Nell'esempio seguente possiamo vedere un blocco modulare e i suoi parametri di ingresso e di uscita, così come i tipi di dati di ingresso, questi dati possono essere di tipo Stringa (stringa di caratteri) o Integer (intero o decimale). Mostriamo come viene utilizzato e lo configuriamo per il suo corretto funzionamento.



Ogni blocco di modulo avrà la sua descrizione e sarà nominato nel caso in cui abbia qualche dipendenza obbligatoria o opzionale di altri blocchi usati come parametri di input, il processo di integrazione sarà annunciato.

Bloccare per creare una lista di stringhe temporanea in un array predefinito chiamato internamente "catena" - (AddHash).

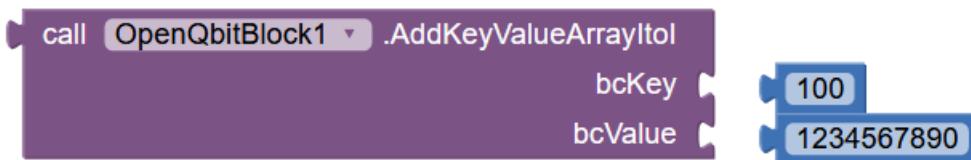


Parametri di ingresso: **blocco** <stringa>

Parametri di uscita: Non applicabile.

Descrizione: Blocco per la memorizzazione di un array temporaneo di hash o stringhe in un array predefinito chiamato internamente "catena".

Blocco per creare array chiave-valore (**Integer-Integer**) - (AddKeyValueArrayItol)



Parametri di ingresso: **bcKey** <Integer>, bcValue <Integer>, bcValue

Parametri di uscita: Restituisce i valori immessi all'ingresso.

Descrizione: E' una disposizione temporanea del valore della chiave, è predefinita con nome interno "Itol_UTXO" questo è utile per elaborare le transazioni UTXO.

Blocco per creare array chiave-valore (**Integer-String**) - (AddKeyValueArrayItoS)

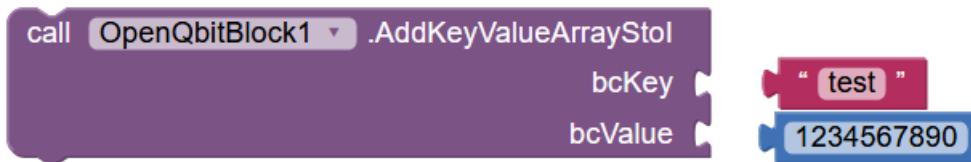


Parametri di ingresso: **bcKey** <Integer>, bcValue < Stringa >

Parametri di uscita: Restituisce i valori immessi all'ingresso.

Descrizione: E' una disposizione temporanea del valore della chiave, è predefinita con il nome interno "ItoS_UTXO", utile per elaborare le transazioni UTXO.

Blocco per creare array di valori chiave (**String-Integer**) - (**AddKeyValueArrayItol**)



Parametri di ingresso: **bcKey** <Stringa>, **bcValue** <Integer>

Parametri di uscita: Restituisce i valori immessi all'ingresso.

Descrizione: Si tratta di una disposizione temporanea del valore della chiave, è predefinita con il nome interno "**Stol_UTXO**", questo è utile per elaborare le transazioni UTXO.

Blocco per creare array chiave-valore (**Stringa-Stringa**) - (**AddKeyValueArrayStoS**)

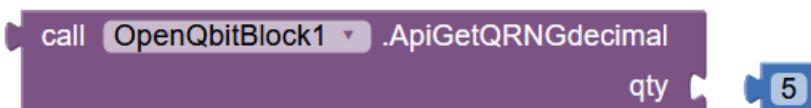


Parametri di ingresso: **bcKey** <String>, **bcValue** <String>, **bcValue**

Parametri di uscita: Restituisce i valori immessi all'ingresso.

Descrizione: Si tratta di una disposizione temporanea del valore della chiave, è predefinita con il nome interno "**StoS_UTXO**", questo è utile per elaborare le transazioni UTXO.

Blocco per la generazione di numeri quantistici casuali decimali - (**ApiGetQRNGdecimal**)



Parametri di ingresso: **qty** <Intero>

Parametri di uscita: Dà la quantità "qty" di numeri decimali quantistici casuali inseriti nei numeri in ingresso sono all'interno dell'intervallo 0 e 1 nel formato JSON.

Esempio:

qty = 5; uscita: {"risultato": [0,5843012986202495, 0,7746497687824652, 0,05951126805960929, 0,1986079055812694, 0,036897834343989999279]}

Descrizione: Generatore di numeri casuali quantistici (QRNG) API

Blocco per la generazione di numeri quantistici casuali decimali - (**ApiGetQRNGdecimale**)



Parametri di ingresso: **qty <Integer>**, min <Integer>, max <max>

Parametri di uscita: Dà la quantità "qty" di numeri interi quantistici casuali inseriti nell'ingresso i numeri si trovano nell'intervallo di min e max in formato JSON.

Esempio:

qty = 8, min = 1, max = 100; uscita: {"risultato": [3, 53, 11, 2, 66, 44, 9, 78]}

Descrizione: Generatore di numeri casuali quantistici (QRNG) API

Blocco Hash "SHA256" per stringhe di caratteri (**ApplySha256**).



Parametri di ingresso: **ingresso <Stringa>**

Parametri di uscita: Calcola l'hash "SHA256" di una stringa di caratteri.

Esempio:

Ingresso = "Mini BlocklyChain è modulare".

uscita: f41af7e61c3b02fdd5e5c612302b62a2dd52fc38f9de97cb2afd827e8804db8

Descrizione: Funzione di rimozione dell'hash "SHA256". Sha o SHA si riferiscono a: Secure Hash Algorithm, un insieme di funzioni hash progettate dalla United States National Security Agency. L'SHA256 utilizza un algoritmo a 256 bit.

Blocco per pulire la "catena" della matrice interna predefinita (**ClearBlockList**).

call **OpenQbitBlock1** ▾ **.ClearBlockList**

Parametri di ingresso e di uscita: Non applicabile

Descrizione: Cancellare tutti gli elementi che hanno la disposizione temporale interna "a catena".

Blocco per pulire l'array interno predefinito (**Integer-Integer**) - (**ClearItol**).

call **OpenQbitBlock1** ▾ **.ClearItol**

Parametri di ingresso e di uscita: Non applicabile

Descrizione: Cancellare tutti gli elementi che hanno la disposizione temporanea interna "Itol_UTXO".

Blocco per pulire l'array interno predefinito (**Integer-String**) - (**ClearItoS**).

call **OpenQbitBlock1** ▾ **.ClearItoS**

Parametri di ingresso e di uscita: Non applicabile

Descrizione: Cancellare tutti gli elementi che hanno la disposizione temporanea interna "ItoS_UTXO".

Blocco per pulire l'array interno predefinito (**String-Integer**) - (**ClearStol**).

call **OpenQbitBlock1** ▾ **.ClearStol**

Parametri di ingresso e di uscita: Non applicabile

Descrizione: Cancellare tutti gli elementi che hanno la disposizione temporanea interna "Stol_UTXO".

Blocco per pulire l'array interno predefinito (**String-String-String**) - (**ClearStoS**).

call **OpenQbitBlock1** ▾ **.ClearStoS**

Parametri di ingresso e di uscita: Non applicabile

Descrizione: Cancellare tutti gli elementi che hanno la disposizione temporanea interna "StoS_UTXO".

Blocco per decodificare una stringa su Base10. (**DecodeBase58**)



Parametri di ingresso: **ingresso<Stringa>**

Parametri di uscita: fornisce la stringa originale che è stata utilizzata nel blocco (**EncodeBase58**).

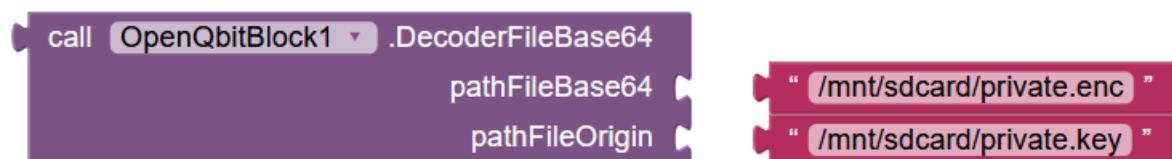
Esempio:

Ingresso = oBRN8Aj67eJsbRGHfNSF9PTdZYGandVWrwMW7mTX

Uscita: "Mini BlocklyChain è modulare".

Descrizione: Converte una stringa Base58 nel testo originale riportato nel blocco (**EncodeBase58**)

Blocco per decodificare un file con algoritmo Base64 (**DecoderFileBase64**).



Parametri di ingresso: **pathFileBase64 <String>** , **pathFileOrigin <String>** , **pathFileOrigin**

Parametri di uscita: File sorgente che è stato inserito nel blocco (**EncoderFileBase64**)

Descrizione: Un file Base64 viene convertito nel file originale che è stato inserito nel blocco (**EncoderFileBase64**).

Bloccare per sapere se l'array interno predefinito "catena" è vuoto. (**EmptyBlockList**)



Parametri di ingresso: Non applicabile.

Parametri di uscita: Restituisce "True" se vuoto o restituisce "False" se si dispone di dati.

Descrizione: Bloccare per chiedere se l'array interno temporaneo predefinito "a catena" ha elementi.

Blocco per codificare una stringa di caratteri su Base58. (**EncodeBase58**).



Parametri di ingresso: **ingresso<Stringa>**

Parametri di uscita: fornisce la stringa originale che è stata utilizzata nel blocco (**EncodeBase58**).

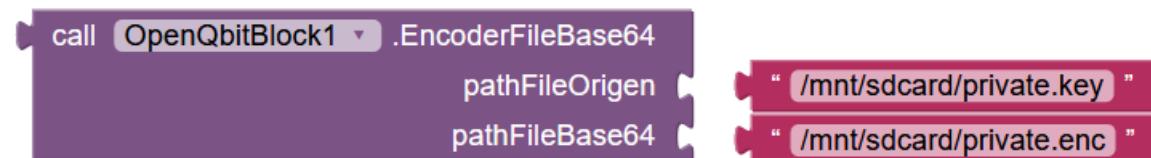
Esempio:

Ingresso = "Mini BlocklyChain è modulare".

La nostra produttività: oBRN8Aj67yJsbRGHfNSF9PTdZYGyVWrwMW7mTX

Descrizione: Converte una catena al seno in una catena in Base58. L'algoritmo Base58 è un gruppo di schemi di codifica da binario a testo usati per rappresentare grandi numeri interi come testo alfanumerico, introdotti da Satoshi Nakamoto per l'uso con Bitcoin.

Blocco per la codifica di un file con algoritmo Base64 (**EncoderFileBase64**).

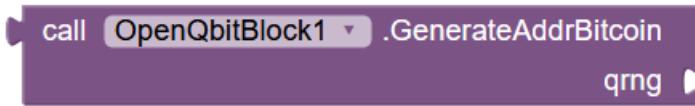


Parametri di ingresso: **pathFileOrigin <String>** , **pathFileBase64 <String>** , **pathFileBase64 <String>**

Parametri di uscita: File codificato Base64.

Descrizione: Converte un file sorgente di qualsiasi formato in un file Base64. I nomi dei file possono essere arbitrari e scelti dall'utente.

Generatore di blocchi di indirizzi utente (**GenerateAddrBitcoin**).



Unità obbligatoria: Block (**ApiGetQRNGinteger**),

Dipendenze (opzionale): estensione **OpenQbitFileEncription**, estensione **OpenQbitFileDecryption** e **OpenQbitSQLite**.

Parametri di ingresso: **qrng** < dipendenza obbligatoria>

Parametri di uscita: indirizzo di transazione alfanumerico a 34 caratteri e indirizzo di utilizzo keyStore

Descrizione: Blocco per creare un nuovo indirizzo di transazione generico Bitcoin per l'utente e generatore di chiavi private (Firma digitale per l'invio di transazioni) e chiave pubblica (Indirizzo pubblico per effettuare transazioni). Questo generatore di chiavi è fondamentalmente il generatore di indirizzi da utilizzare in un portafoglio digitale o comunemente chiamato "Portafoglio".

Questo blocco viene usato insieme ai blocchi Quantum Random Number Generator (QRNG) e i due parametri di uscita devono essere inseriti nel KeyStore.

KeyStore è un database che memorizza le chiavi private in formato esadecimale:

Esempio di indirizzo esadecimale memorizzato nel KeyStore

024C8E0501018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

Questa base è usata solo dall'utente locale e le informazioni sono criptate, questo processo avviene attraverso l'uso dell'estensione **OpenQbitSQLite** e delle estensioni di crittografia delle informazioni **OpenQbitAESEncryption** e **OpenQbitAESDecryption**.

Per creare un KeyStore vedere l'appendice "Creazione di un KeyStore". Gli indirizzi generati utilizzano lo stesso algoritmo di indirizzo Bitcoin, con l'identificatore di indirizzo bitcoin iniziale "1".

Gli indirizzi generati dal blocco (**GenerateAddrBitcoin**) sono 34 caratteri alfanumerici composti da 33 caratteri alfanumerici e 1 dell'identificatore Bitcoin come segue:

12dRugNcdxK39288NjcDV4GX7rMsKCGn6k

Blocco generatore di indirizzi utente (**GenerateAddrEthereum**).



Dipendenza obbligatoria: ottenere un gettone all'indirizzo:
<https://accounts.blockcypher.com/signup>

Dipendenze (opzionale): estensione **OpenQbitFileEncryption**, estensione **OpenQbitFileDecryption** e **OpenQbitSQLite**.

Parametri di ingresso: **token** < dipendenza obbligatoria - Stringa>

Parametri di uscita: l'indirizzo di transazione a 40 caratteri alfanumerici non include l'indicatore iniziale Ethereum "0x" che ci darebbe i 42 caratteri di un indirizzo comune. Restituisce anche la chiave pubblica e la chiave privata.

Esempio:

```
{  
  "privato": "227ac59f480131272003c2d723a7795ebd3580acaab62b5c537989e2ce4e08ef",  
  "pubblico":  
    "04e2d55ebccd32a7384e096df559cc36b856c64a16e5b402e10585dc3ea055672aafa84df8  
    a859531570a650a8ab1e7a22949100efa1aa5f072c035551cac1ce",  
  "indirizzo": "14e150399b0399f787b4d6fe30d8b251375f0d66"}
```

Descrizione: Blocco per creare un nuovo indirizzo di transazione per l'utente e il generatore di chiavi private (firma digitale per inviare transazioni) e chiave pubblica (indirizzo pubblico per effettuare transazioni). Questo generatore di chiavi è un'API esterna ed è necessario avere una connessione Wifi o mobile, è fondamentalmente il generatore di indirizzi per usarlo in un portafoglio digitale o comunemente chiamato "Portafoglio".

È possibile creare aKeyStore è una banca dati che memorizza le chiavi private in formato esadecimale, vedi Appendice "Creazione di KeyStore".

Esempio di indirizzo esadecimale memorizzato nel KeyStore

0x14e150399b0399f787b4d6fe30d8b251375f0d66

La chiave privata è usata solo dall'utente locale e le informazioni sono criptate, questo processo avviene attraverso l'uso dell'estensione **OpenQbitSQLite** e delle estensioni di crittografia delle informazioni **OpenQbitAESEncryption** e **OpenQbitAESDecryption**.

Blocco generatore di indirizzi utente (**GenerateAddrMiniBlocklyChain**).



Unità

obbligatoria: Block ([ApiGetQRNGinteger](#)),

Dipendenze (opzionale): estensione [OpenQbitFileEncription](#), estensione [OpenQbitFileDecryption](#) e [OpenQbitSQLite](#).

Parametri di ingresso: **qrng** < dipendenza obbligatoria>

Parametri di uscita: indirizzo della transazione a 36 caratteri alfanumerici e indirizzo di utilizzo del keyStore. Metodo del blocco di revisione ([PairKeysMBC](#)).

Descrizione: Blocco per creare un nuovo indirizzo di transazione per l'utente e il generatore di chiavi private (firma digitale per inviare transazioni) e chiave pubblica (indirizzo pubblico per effettuare transazioni). Questo generatore di chiavi è fondamentalmente il generatore di indirizzi per utilizzarlo in un portafoglio digitale o comunemente chiamato "Portafoglio".

Questo blocco viene usato insieme ai blocchi Quantum Random Number Generator (QRNG) e i due parametri di uscita devono essere inseriti nel KeyStore.

KeyStore è un database che memorizza le chiavi private in formato esadecimale:

Esempio di indirizzo esadecimale memorizzato nel KeyStore

024C8E0501018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

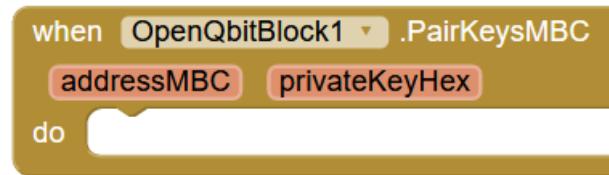
Questa base è usata solo dall'utente locale e le informazioni sono criptate, questo processo avviene attraverso l'uso dell'estensione [OpenQbitSQLite](#) e delle estensioni di crittografia delle informazioni [OpenQbitAESEncryption](#) e [OpenQbitAESDecryption](#).

Per creare un KeyStore vedere l'appendice "Creazione di un KeyStore". Gli indirizzi generati utilizzano lo stesso algoritmo di indirizzo bitcoin, l'unica differenza è che l'identificatore di indirizzo bitcoin che è "1" o "3" viene cambiato con il Mini BlocklyChain identifier "MBC".

Gli indirizzi generati dal blocco (**GenerateAddrMiniBlocklyChain**) sono 36 caratteri alfanumerici e sono composti da 33 caratteri alfanumerici e 3 delle lettere maiuscole dell'identificatore Mini BlocklyChain "MBC" come segue:

Mbc2dRugNcdxK39288NjcDV4GX7rMsKCGn6k

Il metodo a blocchi (**PairKeysMBC**) è l'uscita a blocchi (**GenerateAddrMiniBlocklyChain**).



Parametri di ingresso: Non applicabile.

Parametri di uscita: **addressMBC <String>**, **privateKeyHex <String>**.

Descrizione: Indirizzo utente pubblico in formato Mini BlocklyChain e chiave privata in formato esadecimale.

esempio:

indirizzoMBC: MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k

privateKeyHex:

024C8E0501018319ARD4BBB04E184C307BFF115976A05F974C7D945B5151E490ERD

Generatore di blocchi di indirizzi utente (**GenerateKeyValuePair**).



Unità obbligatoria: Block (**ApiGetQRNGinteger**),

Dipendenze (opzionale): estensione **OpenQbitFileEncryption**, estensione **OpenQbitFileDecryption** e **OpenQbitSQLite**.

Parametri di ingresso: **qrng < dipendenza obbligatoria>**

Parametri di uscita: Fornisce la chiave pubblica e la chiave primaria per l'utente locale

Descrizione: Genera chiavi pubbliche e primarie utilizzando l'algoritmo ECC (1) e con formato esadecimale.

- (1) La crittografia delle curve ellittiche (ECC) è una variante della crittografia asimmetrica o a chiave pubblica basata sulla matematica delle curve ellittiche.

Blocco delle transazioni di firma (**GenerateSignature**).

```
call OpenQbitBlock1 .GenerateSignature
```

Dipendenza richiesta: Block (**GenerateKeyPairs**), Block (**SenderLoadKeyPair**), Block (**RecipientLoadKeyPair**). Prefazionare questi blocchi prima dell'uso.

Parametri di ingresso: < Dipendenze di controllo obbligatorie>

Parametri di uscita: Nessuna uscita.

Descrizione: genera una firma digitale dal Mittente applicata al bene inviato nella transazione al Destinatario, questa firma sarà confermata quando la transazione viene elaborata da qualche nodo della rete, con questa firma il sistema Mini BlocklyChain assicura che il bene non sia stato modificato in esso inviato e che sia conforme al rapporto mittente-ricevente e non sia deviato o applicato ad un altro indirizzo digitale.

Bloccare per ottenere il saldo totale delle attività di un utente (**GetBalance**).

```
call OpenQbitBlock1 .GetBalance  
fromAddrMBC " MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k "
```

Dipendenza obbligatoria: Block (**ConnectorTransactionTail**).

Parametri di ingresso: **fromTransTail <Array String>**, < Dipendenze delle schede obbligatorie>

Parametri di uscita: controlla le spese in entrata e in uscita per ogni transazione.

Descrizione: verifica il saldo per approvare se l'utente ha beni da inviare o se i beni che riceve vengono aggiunti al suo saldo.

Bloccare per ottenere q22 dal cellulare. (**GetDeviceID**)

```
call OpenQbitBlock1 .GetDeviceID
```

Parametri di ingresso: **Non applicabile**

Parametri in uscita: fornisce l'identificativo interno del cellulare.

Descrizione: Fornisce l'identificatore di telefono cellulare IMEI unico per ogni dispositivo.

Bloccare per rimuovere l'hash del RIPEMD-160 da una stringa. (**Ripemd-160**)



Parametri di ingresso: **str <Stringa>**

Parametri di uscita: Calcola l'hash "RIPEMD-160" di una stringa di caratteri.

Esempio:

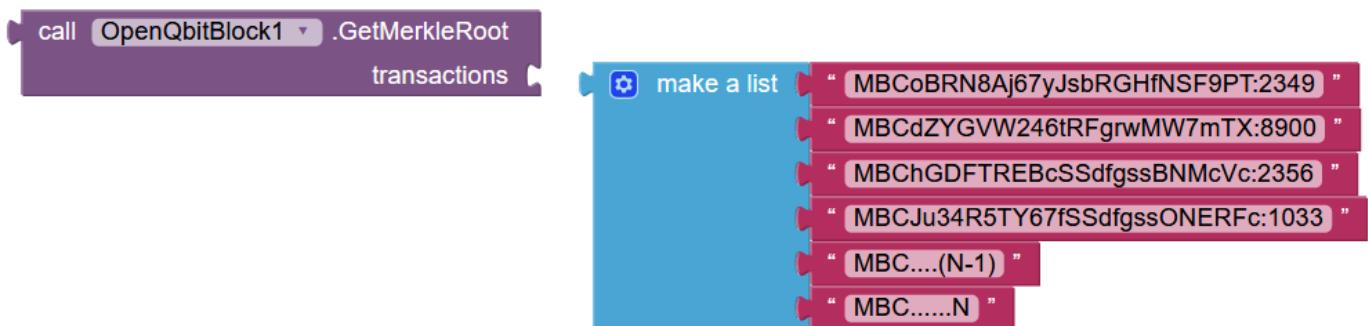
Ingresso = "Mini BlocklyChain è modulare".

uscita: ae29436e4b8b8ea8ed6143f3f92380dfa2f4f47336

Descrizione: Ottenere l'hashish "RIPEMD-160". Questo hash viene utilizzato nel processo di generazione di un indirizzo valido per Bitcoin e Mini BlocklyChain.

RIPEMD-160 (acronimo di RACE Integrity Primitives Evaluation Message Digest) è un algoritmo di digest dei messaggi a 160 bit (e funzione di hash crittografico).

Blocco per calcolare l'albero di Merkler. (**GetMerkleRoot**)



Parametri di ingresso: **transazioni <Array String>**

Parametri di uscita: Fornisce un tipo di hashish "SHA256".

Esempio:

Input = coda di transazione, una disposizione di tutte le transazioni da elaborare.

uscita: b4a44c42b6070825f763cd118d6ab49a8e80bbb7cdc0225064f8e042b94196bd

Descrizione: Ottenere l'hash "SHA256" utilizzando l'algoritmo ad albero di Merkle

Un Merkle Hash Tree è una struttura ad albero di dati binari o non binari in cui ogni nodo che non è una foglia è etichettato con l'hash della concatenazione delle etichette o dei valori (per

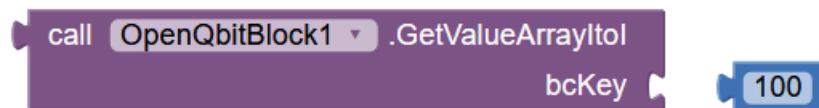
i nodi fogliacei) dei suoi nodi figli. Si tratta di una generalizzazione delle liste di hash e delle stringhe di hash.

Consente di collegare un gran numero di dati separati ad un unico valore di hash, l'hash del nodo radice dell'albero. Ciò fornisce un metodo sicuro ed efficiente per verificare il contenuto di grandi strutture di dati. Nelle sue applicazioni pratiche, l'hash del nodo radice è solitamente firmato per garantire la sua integrità e la totale affidabilità della verifica. Dimostrare che un nodo fogliaceo fa parte di un dato albero di hashish richiede una quantità di dati proporzionale al logaritmo del numero di nodi dell'albero.

Attualmente l'uso principale degli alberi Merkle è quello di rendere sicuri i blocchi di dati ricevuti da altri peer-to-peer nelle reti peer-to-peer, per garantire che siano ricevuti senza danni e senza essere alterati.

Serve a verificare che una coda che ha le transazioni da elaborare non sia stata modificata e a garantirne l'integrità per la dispersione in tutti i nodi della rete Mini BlocklyChain. Tutti i nodi devono eseguire questo algoritmo per garantire l'integrità di ogni transazione che verrà applicata.

Bloccare per ottenere un valore dell'array predefinito "Itol_UTXO" (**GetValueArrayItol**).

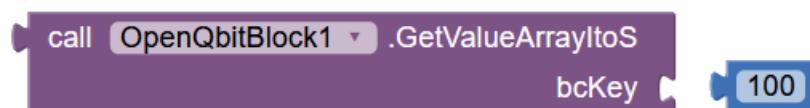


Parametri di ingresso: **bcKey** <Integer>

Parametri di uscita: Restituisce il valore <Integer> memorizzato nell'etichetta con il numero dato come input.

Descrizione: Si tratta di una richiesta alla disposizione temporanea del valore della chiave, che è predefinita con il nome interno "Itol_UTXO", utile per elaborare le transazioni UTXO.

Bloccare per ottenere un valore dall'array predefinito "ItoS_UTXO" (**GetValueArrayItoS**).



Parametri di ingresso: **bcKey** <Integer>

Parametri di uscita: Restituisce il valore <Stringa> memorizzato nell'etichetta con il numero dato come input.

Descrizione: Si tratta di una richiesta alla disposizione temporanea del valore della chiave, che è predefinita con il nome interno "**ItoS_UTXO**", utile per elaborare le transazioni UTXO.

Bloccare per ottenere un valore dall'array predefinito "StoI_UTXO" (**GetValueArrayStoI**).



Parametri di ingresso: **bcKey** < Stringa

Parametri di uscita: Restituisce il valore <Integer> memorizzato nell'etichetta con il nome dato come input.

Descrizione: Si tratta di una richiesta alla disposizione temporanea del valore della chiave, che è predefinita con il nome interno "**StoI_UTXO**", utile per elaborare le transazioni UTXO.

Bloccare per ottenere un valore dall'array predefinito "StoS_UTXO" (**GetValueArrayStoS**).



Parametri di ingresso: **bcKey** < Stringa

Parametri di uscita: Restituisce il valore <Stringa> memorizzato nell'etichetta con il nome dato come input.

Descrizione: Si tratta di una richiesta alla disposizione temporanea del valore della chiave, che è predefinita con il nome interno "**StoS_UTXO**", utile per elaborare le transazioni UTXO.

Validatore di blocco della catena di blocco. (**IsChainValid**)

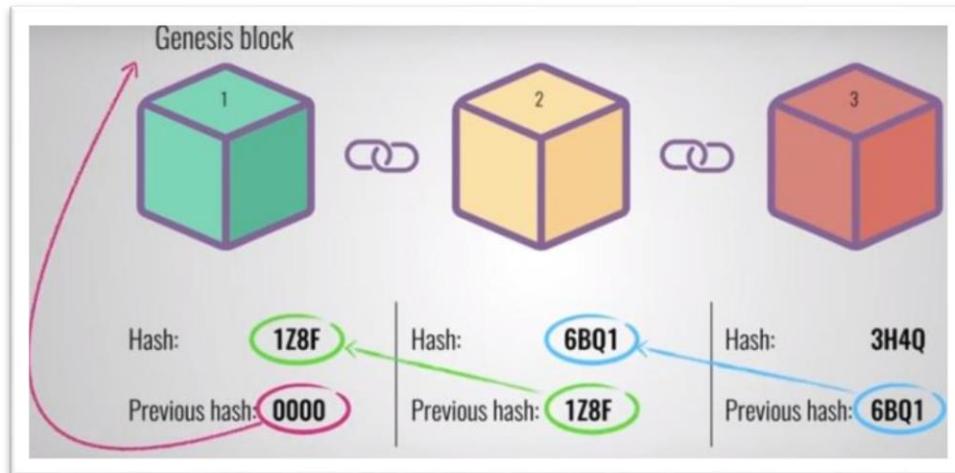


Dipendenza richiesta: Block (**GenerateKeyPairs**), Block (**SenderLoadKeyPair**), Block (**RecipientLoadKeyPair**), Block (**GenerateSignature**). Prefazionare questi blocchi prima dell'uso.

Parametri di ingresso: Non applicabile.

Parametri di uscita: Consegna "True" se la validazione della stringa di blocco è corretta o "False" nel caso la validazione fallisca.

Descrizione: Ci fornisce la validazione dei componenti che sono stati precedentemente inseriti nel sistema a catena a blocchi, questa validazione è la più importante di tutto il sistema. Come funziona la convalida della catena a blocchi o come è comunemente conosciuta in modo generico (BlockChain). È la parte centrale di ogni sistema.



Come si vede nell'immagine precedente, ogni blocco aggiunto nel sistema di memorizzazione è collegato al blocco precedente attraverso gli algoritmi di hash.

Per esempio, quando si crea un nuovo blocco da aggiungere, l'hash che rappresenta le nuove informazioni si ottiene prendendo l'hash del nuovo blocco di informazioni + l'hash precedente. Con questo tipo di collegamento viene rilevata qualsiasi minima modifica nella memorizzazione delle catene a blocchi che consente una sicurezza dei dati molto elevata ed efficace.

Di seguito un esempio di come una stringa di blocchi viene memorizzata in un database SQLite, vedremo come l'hash precedente è collegato al nuovo hash dell'ultimo blocco (ultima coda di transazioni elaborate) che è stato inserito. Ogni hash in entrambe le colonne "prevhash" e "newhash" rappresenta le informazioni della coda delle transazioni che sono state elaborate in quel momento.

SQLite Expert Personal 5.3 (x64)

File View Database Object SQL Transaction Tools Help

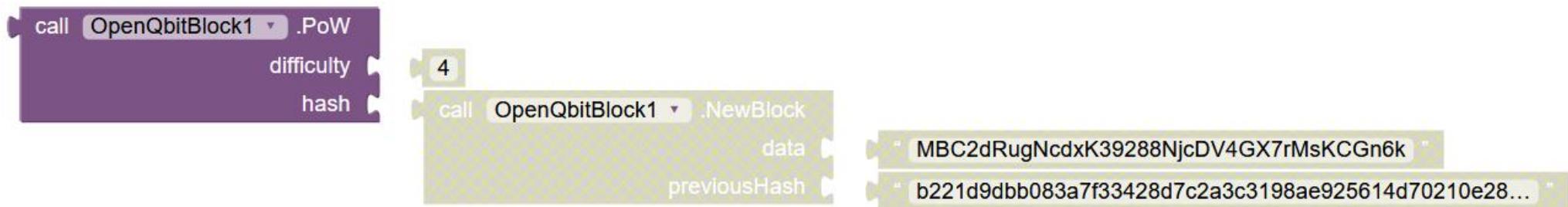
Database: miniblock Table: nblock File: C:\memo\thunkable\miniblock.db

miniblock

rowid	id	prevhash	newhash	opera	trans	balan
1	1	0767c864cef0334f27473902eb9868e7	bdc9065d20a4cd037bb1a7538486403e	2	0	0
2	2	bdc9065d20a4cd037bb1a7538486403e	6619f4809d73a267a4b9ac554bb4523a	1	5	5
3	3	6619f4809d73a267a4b9ac554bb4523a	4d186321c1a7f0f354b297e8914ab240	1	5	5

L'hash precedente è legato al nuovo hash.

Blocco per eseguire il PoW Work Test e per estrarre nuovi blocchi. (**Pow**)



Dipendenza obbligatoria: **NewBlock**. Prima di utilizzarlo, prefacciare questo blocco.

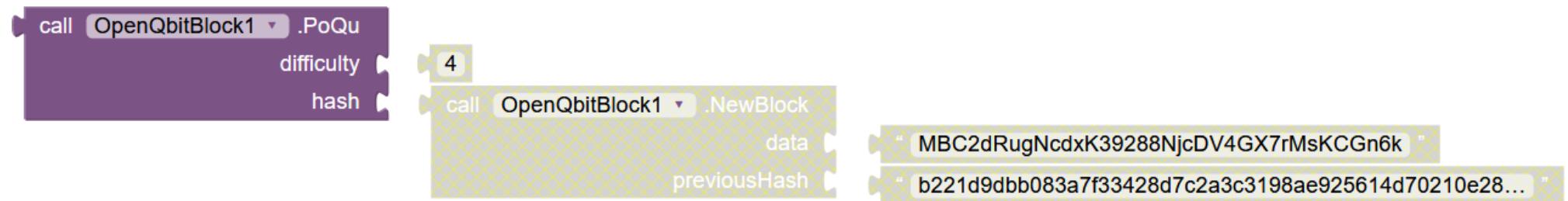
Parametri di ingresso: **difficoltà** <Integer>, hash <dipendenza obbligatoria>

Parametri di uscita: Dà come uscita un hash "SHA256" composto con il #Numero di "zeri" dato nella difficoltà del parametro di ingresso.

Descrizione: Questo blocco svolge un'attività chiamata "estrarre" un nuovo blocco è quello che abbiamo descritto in precedenza come il processo di PoW (Proof of Work), vedi sezione Cos'è la Proof of Quantum (PQu)?

L'estrazione o la gestione del PoW è un concetto in cui ai nodi viene dato il compito di risolvere un puzzle matematico. Chi lo risolve per primo nel caso di Mini BlocklyChain ha l'opportunità di continuare il processo per essere scelto per essere il vincitore di poter elaborare la coda delle transazioni in attesa di essere elaborate.

Blocco per eseguire il PoW Work Test e per estrarre nuovi blocchi. (**PoQu**)



Dipendenza obbligatoria: **NewBlock**. Prima di utilizzarlo, prefacciare questo blocco.

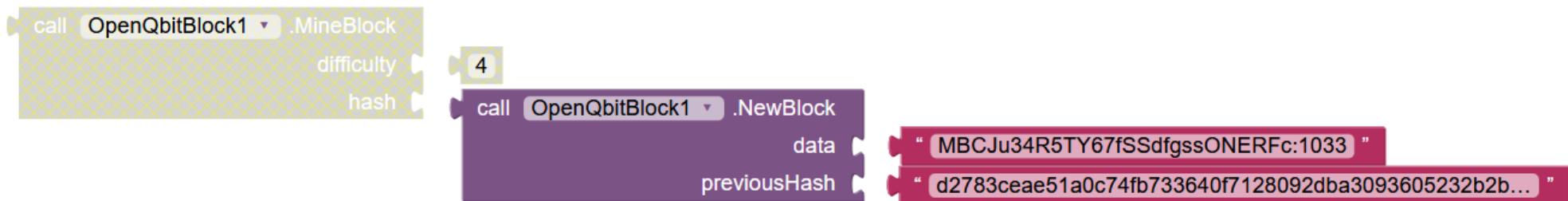
Parametri di ingresso: **difficoltà** <Integer>, hash <dipendenza obbligatoria>

Parametri di uscita: Risultati nel numero "Magic Number" e nel numero quantistico casuale che si trova nell'intervallo 0 e 1 di tipo decimale a 16 cifre, esempio **(0.5843012986202495)** e un hash "SHA256" composto con il numero #Numero di "zeri" dato nella difficoltà del parametro di ingresso.

Descrizione: Questo blocco esegue il processo di "estrazione" di un nuovo blocco è quello che abbiamo descritto in precedenza come il processo di PoW (Proof of Work), ma questo processo chiama la funzione di numero quantico casuale generato dopo il calcolo del numero "nonce" adatto a soddisfare il livello di difficoltà che può essere nell'intervallo minimo 1, massimo 5. Vedi sezione Che cos'è la Proof of Quantum (PQu - Proof Quantum)?

L'estrazione o l'esecuzione del PoW o PoQu è un concetto in cui ai nodi viene dato il compito di risolvere un puzzle matematico. Il nodo che lo risolve per primo nel caso di un qualsiasi sistema a catena di blocchi ha l'opportunità di continuare il processo per essere scelto per essere il vincitore di poter elaborare la coda di transazioni che è in attesa di essere elaborata.

Blocco per la creazione di **nuovi** blocchi (**NewBlock**).

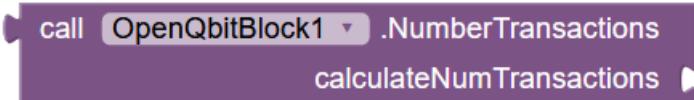


Parametri di ingresso: **dati** <Stringa>, **precedenteHash** <Stringa>

Parametri di uscita: Fornisce un hash calcolato come: **SHA256 (dati (parametri di ingresso) + previousHash (parametro di ingresso) + IMEI (parametro interno) + MerkleRoot (parametro interno))**.

Descrizione: Questo blocco esegue la creazione di un nuovo blocco da elaborare. Esso fornisce come output un hash "SHA256" composto dalla stringa di parametri di input nel caso di dati variabili a seconda di ogni progetto di sistema e l'hash precedente si basa sull'hash della precedente stringa di transazione già elaborata e memorizzata nel sistema di stringa a blocchi Mini BlocklyChain, questi due sono parametri variabili, ci sono due parametri interni al sistema che sono fissi e garantiscono l'integrità delle informazioni e del sistema, questi due parametri interni sono l'ID univoco del telefono cellulare e l'hash dell'albero merklet della coda di transazione che viene elaborata.

Blocco delle transazioni locali totali del nodo (**NumberTransactions**)



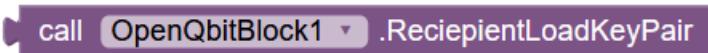
Dipendenza obbligatoria: Block (**AddKeyValueToArrayStoS**). Prima di utilizzarlo, prefacciare questo blocco.

Parametri di ingresso: < Dipendenza obbligatoria>.

Parametri di uscita: Restituisce il totale delle transazioni locali al nodo.

Descrizione: Fornisce il totale delle transazioni che saranno applicate solo ai conti locali del nodo.

Bloccare per leggere l'indirizzo del destinatario nel file binario. (**RecipientLoadKeyValuePair**)

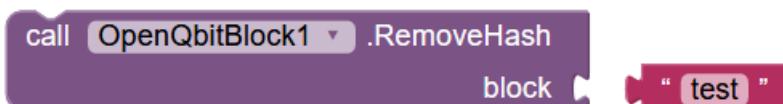


Parametri di ingresso: **Non applicabile**.

Parametri di uscita: Restituisce la chiave privata e la chiave pubblica in formato Base64.

Descrizione: ottiene l'indirizzo privato e pubblico del destinatario da un file binario come parametro di input.

Blocco per cancellare l'elemento nella disposizione temporanea interna "catena" (**RemoveHash**).

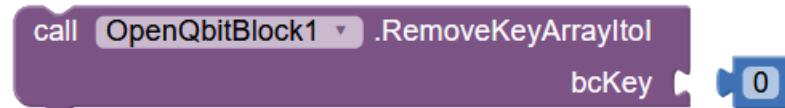


Parametri di ingresso: **blocco < Stringa>**.

Parametri di uscita: Non applicabile.

Descrizione: ottiene l'indirizzo privato e pubblico del destinatario da un file binario come parametro di input.

Blocco per cancellare l'elemento nella disposizione temporanea interna "Itol_UTXO" (**RemoveKeyArrayItol**)

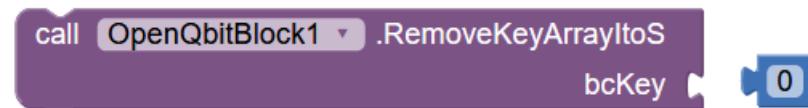


Parametri di ingresso: **bcKey < Integer >**.

Parametri di uscita: Non applicabile, tipo di elemento cancellare <Integer>.

Descrizione: L'elemento associato all'etichetta numerica della disposizione temporanea interna "Itol_UTXO" viene cancellato.

Blocco per cancellare l'elemento nella disposizione temporanea interna "ItoS_UTXO" (**RemoveKeyArrayItoS**).

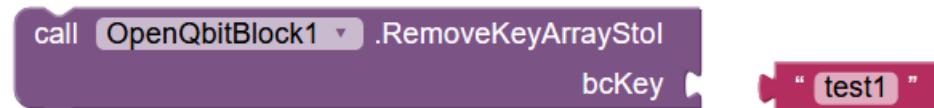


Parametri di ingresso: **bcKey < Integer >**.

Parametri di uscita: Non applicabile, tipo di elemento da cancellare <Stringa>.

Descrizione: L'elemento associato all'etichetta numerica della disposizione temporanea interna "ItoS_UTXO" viene cancellato.

Blocco per la cancellazione dell'elemento nella disposizione temporanea interna "Stol_UTXO" (**RemoveKeyArrayStol**)

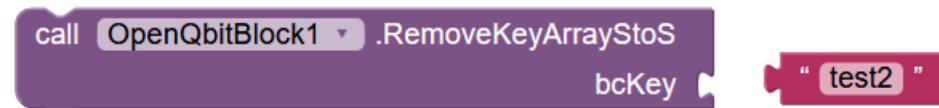


Parametri di ingresso: **bcKey < String >**.

Parametri di uscita: Non applicabile, tipo di elemento cancellare <Integer>.

Descrizione: L'elemento associato all'etichetta numerica della disposizione temporanea interna "Stol_UTXO" viene cancellato.

Blocco per la cancellazione dell'elemento nella disposizione temporanea interna "StoS_UTXO" (**RemoveKeyArrayStoS**)

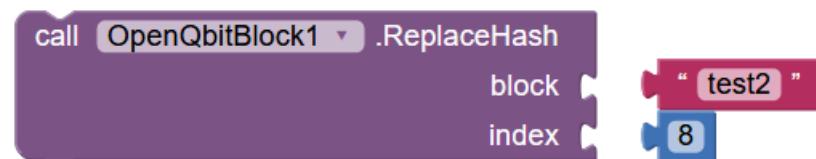


Parametri di ingresso: **bcKey < String>**.

Parametri di uscita: Non applicabile, tipo di elemento da cancellare <Stringa>.

Descrizione: Cancellare l'elemento dell'etichetta di disposizione temporanea interna "StoS_UTXO".

Blocco per sostituire un valore della disposizione temporanea interna "a catena" (**ReplaceHash**).

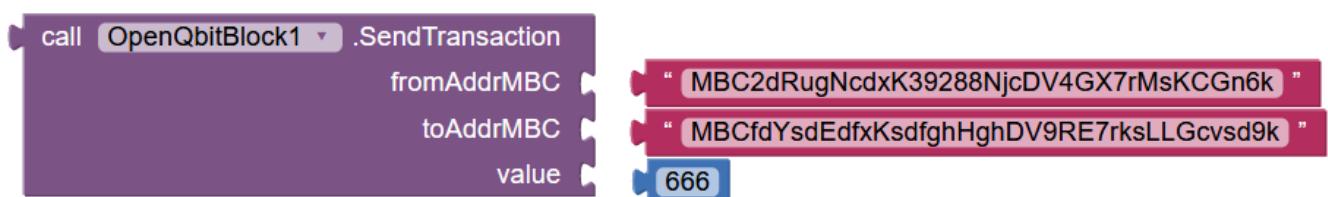


Parametri di ingresso: **blocco < Stringa >, indice < Intero**

Parametri di uscita: Non applicabile.

Descrizione: L'elemento associato all'indice "indice" viene sostituito con il valore dell'etichetta "blocco" nella disposizione interna temporanea "catena".

Blocca per avviare (**inviare**) una nuova transazione (**SendTrasaction**).



Parametri di ingresso: **daAddrMBC < Stringa >, aAddrMBC < Stringa >, valore < Numero intero**

Parametri di uscita: Restituisce il valore "True" se la transazione è stata inviata con successo o "False" se non è stata inviata con successo.

Descrizione: Blocco che trasforma l'indirizzo del mittente e del destinatario in formato binario e che viene allegato alla coda delle transazioni da elaborare.

Bloccare per leggere l'indirizzo del mittente nel file binario. (**SenderLoadKeyPair**)

 call OpenQbitBlock1 .SenderLoadKeyPair

Parametri di ingresso: **Non applicabile**.

Parametri di uscita: Restituisce la chiave privata e la chiave pubblica in formato Base64.

Descrizione: ottiene l'indirizzo privato e pubblico del mittente da un file binario come parametro di input.

Blocco per ottenere il numero dell'elemento della disposizione temporanea "catena" (**SizeBlockList**).

 call OpenQbitBlock1 .SizeBlockList

Parametri di ingresso: **Non applicabile**.

Parametri di uscita: Restituisce il numero dell'elemento della "catena" della matrice interna.

Descrizione: Riceve il numero dell'elemento della "catena" della matrice interna.

Blocco per ottenere il numero dell'elemento della disposizione temporanea "Itol_UTXO" (**Sizeltol**)

 call OpenQbitBlock1 .Sizeltol

Parametri di ingresso: **Non applicabile**.

Parametri di uscita: restituisce il numero dell'elemento della matrice interna "Itol_UTXO".

Descrizione: ottiene il numero dell'elemento dell'array interno "Itol_UTXO".

Blocco per ottenere il numero dell'elemento della disposizione temporanea "ItoS_UTXO" (**SizeltoS**)

 call OpenQbitBlock1 .SizeltoS

Parametri di ingresso: **Non applicabile**.

Parametri di uscita: restituisce il numero dell'elemento della matrice interna "ItoS_UTXO".

Descrizione: ottiene il numero dell'elemento dell'array interno "ItoS_UTXO".

Blocco per ottenere il numero dell'elemento della disposizione temporanea "Stol_UTXO" (SizeStol)



Parametri di ingresso: **Non applicabile**.

Parametri di uscita: restituisce il numero dell'elemento della matrice interna "Stol_UTXO".

Descrizione: ottiene il numero dell'elemento della matrice interna "Stol_UTXO".

Blocco per ottenere il numero dell'elemento della disposizione temporanea "StoS_UTXO" (SizeStoS)

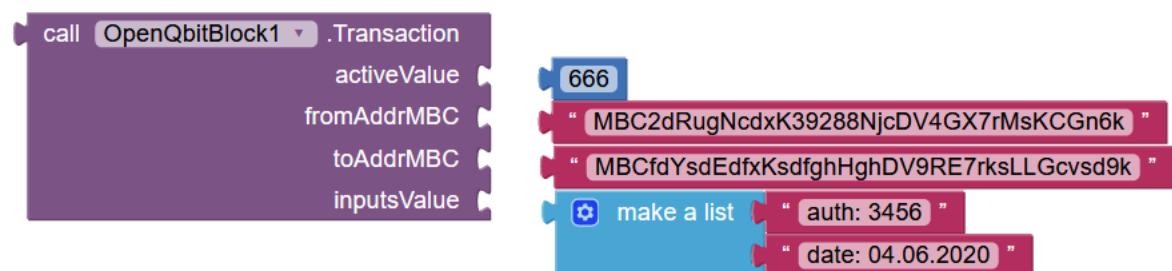


Parametri di ingresso: **Non applicabile**.

Parametri di uscita: restituisce il numero dell'elemento della matrice interna "StoS_UTXO".

Descrizione: ottiene il numero dell'elemento della matrice interna "StoS_UTXO".

Blocco della creazione di transazioni con valori aggiuntivi (**Transaction**).



Parametri di ingresso: **activeValue < Integer>**, **daAddrMBC < String>**, **aAddrMBC < String>**, **inputsValue < Array String>**.

Parametri di uscita: ci dà, gli indirizzi in formato binario e il valore inpusetValue convertito in una stringa 'String', e ci dà l'hash "SHA256" del valore ActiveValue.

Descrizione: Prepara una nuova transazione che deve essere elaborata dai nodi.

Mini blocco di convalida indirizzi BlocklyChain (**ValidateAddMiniBlocklyChain**)



Parametri di ingresso: Indirizzi utente in formato Mini BlocklyChain.

Parametri di uscita: Restituisce "True" se l'indirizzo è nel formato corretto o "False" se l'indirizzo non è valido.

Descrizione: Convalida se l'indirizzo Mini BlocklyChain inserito è corretto, questo blocco applica un algoritmo per verificare se l'indirizzo è stato creato utilizzando il meccanismo di creazione dell'indirizzo da utilizzare nel sistema Mini BlocklyChain.

Blocco di convalida degli indirizzi Bitcoin. (**ValidateAddBitcoin**)



Parametri di ingresso: **addr** < String> , indirizzi utente formattati Bitcoin (accetta indirizzi Bitcoin con identificatore iniziale "1", gli indirizzi con identificatore "3" non sono applicabili).

Parametri di uscita: Restituisce "True" se l'indirizzo è nel formato corretto o "False" se l'indirizzo non è valido.

Descrizione: Convalida se l'indirizzo Bitcoin inserito è corretto, questo blocco applica un algoritmo per verificare se l'indirizzo è stato creato utilizzando il meccanismo di creazione dell'indirizzo da utilizzare nel sistema Bitcoin.

Blocco di verifica della firma digitale per la transazione in corso. (**VerifySignature**).



Parametri di ingresso: **Non applicabile**.

Parametri di uscita: Restituisce "True" se il controllo è valido o "False" se il controllo non è valido.

Descrizione: ottiene la verifica della firma digitale che il mittente avrebbe dovuto vedere fatta nel processo di invio della transazione che si desidera effettuare. In questo processo di verifica si verifica che il valore sorgente inviato non sia stato alterato nel canale in cui è stata inviata la transazione, oltre a verificare il destinatario a cui la transazione deve essere applicata

20. Utilizzo di blocchi per database SQLite (versione MiniSQLite)

In questa sezione vedremo come utilizzare i blocchi per eseguire due operazioni principali che ci interessano per la funzionalità del sistema Mini BlocklyChain che è quello di "INSERIRE" i dati nella catena dei blocchi e interrogarli.

NOTA: Le transazioni nel database SQLite sono diverse da quelle inviate dai nodi da applicare nel sistema Mini Blocklychain.

Le transazioni nel database si basano su un modello CRUD (Create, Read, Update and Delete) e sono i processi che possono essere eseguiti con dati esterni o interni solo nel database SQLite. Nei sistemi a catena di blocchi vengono utilizzati principalmente i processi di Creazione e Lettura, per la sicurezza vengono scartati i processi di aggiornamento o cancellazione e più dove è memorizzata la catena di blocchi che dà la sicurezza integrale del sistema.

D'altra parte, le transazioni inviate dai nodi si riferiscono a tutti i processi che comportano l'azione di invio di qualche tipo di asset tra i membri (nodi) del sistema Mini BlocklyChain, questo tipo di transazioni include diversi processi per la sua applicazione, questi possono essere dalla creazione di un indirizzo digitale, una firma digitale, una convalida della firma, un processo all'interno del database SQLite, tra gli altri processi.

Inizieremo con la definizione e l'utilizzo dei blocchi del database SQLite versione MiniSQLite questa versione è integrata solo da 8 blocchi. Avete una versione completa per manipolare i dati nel database SQLite, tuttavia, per scopi pratici è sufficiente il sistema Mini BlocklyChain con la versione MiniSQLite.

Nel caso si voglia rivedere tutti i componenti il loro uso e la loro descrizione si veda l'allegato "Blocchi estesi per database SQLite".

Blocchi di versione MiniSQLite:

Bloccare per avviare un qualche tipo di transazione nel database SQLite (**BeginTransaction**)

call OpenQbitQSQLite1 .BeginTransaction

Unità obbligatorie: Block (**ImportDatabase**), Block (**OpenDatabase**).

Parametri di ingresso: **Utilizzare prima di < Dipendenza(i) obbligatoria(e) >**

Parametri di output: Non applicabile, avvia una transazione in un database SQLite.

Descrizione: Blocco che avvia un processo nel database SQLite, è necessario aver prima eseguito il blocco di importazione del database (**ImportDatabase**) e il blocco aperto del database (**OpenDatabase**).

Blocca per fare Commit in SQLite. (**CommitTrasaction**)

call **OpenQbitQSQLite1** .**CommitTransaction**

Dipendenze richieste: Block (**BeginTransaction**), Block (**ImportDatabase**), Block (**OpenDatabase**).

Parametri di ingresso: **Utilizzare prima di** < Dipendenza(i) obbligatoria(e) >

Parametri di output: Non applicabile, esegue un **commit di una** transazione in un database SQLite.

Descrizione: Blocco che avvia un processo di **commit sul** database SQLite, è necessario aver prima eseguito il blocco di importazione del database (**ImportDatabase**) e il blocco di apertura del database (**OpenDatabase**).

Blocco per chiudere il database SQLite importato o esportato (**CloseDatabase**)

call **OpenQbitQSQLite1** .**CloseDatabase**

Unità obbligatorie: Block (**ImportDatabase**), Block (**OpenDatabase**).

Parametri di ingresso: **Utilizzare prima di** < Dipendenza(i) obbligatoria(e) >

Parametri di uscita: Non applicabile, chiude un database SQLite.

Descrizione: Blocco che chiude il database SQLite, è necessario aver prima eseguito il blocco di importazione del database (**ImportDatabase**) e il blocco di apertura del database (**OpenDatabase**).

Bloccare l'esportazione di un database SQLite (**ExportDatabase**).

call **OpenQbitQSQLite1** .**ExportDatabase**

fileName

“ **mbcExport.sqlite** ”

Unità obbligatorie: Block (**ImportDatabase**), Block (**OpenDatabase**).

Parametri di input: **nome fileName < Stringa>** inserire un percorso dove si può trovare un database già creato in formato SQLite.

Utilizzare prima di < Dipendenza(e) obbligatoria(e) >

Parametri di uscita: Esportazione in un database SQLite.

Descrizione: Blocco che esporta un database SQLite, è necessario aver prima eseguito il blocco di importazione del database (**ImportDatabase**) e il blocco aperto del database (**OpenDatabase**).

Bloccare per importare il database SQLite. (**ImportDatabase**)

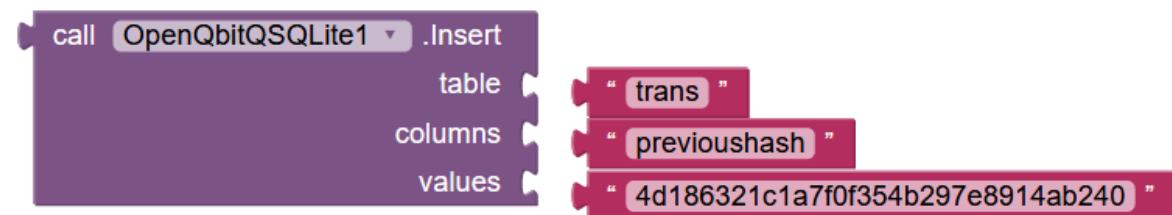


Parametri di input: **nome fileName < Stringa>** inserire un percorso dove si può trovare un database già creato in formato SQLite.

Parametri di output: Avvia un database SQLite per eseguire le transazioni.

Descrizione: Blocco che avvia un processo nel database SQLite, è necessario aver prima eseguito il blocco di importazione del database (**ImportDatabase**) e il blocco aperto del database (**OpenDatabase**).

Bloccare per inserire i dati nel database SQLite. (**Inserire**)



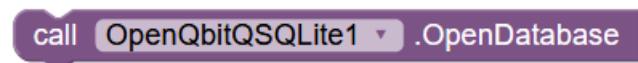
Unità obbligatorie: Block (**ImportDatabase**), Block (**OpenDatabase**).

Parametri di ingresso: **tabella < Stringa>** , colonne < Stringa> , valori < Stringa> , Utilizzare prima < Dipendenza(e) obbligatoria(e) >.

Parametri di uscita: Inserisce una transazione in un database SQLite.

Descrizione: Blocco che inserisce i dati nel database SQLite, è necessario aver prima eseguito il blocco di importazione del database (**ImportDatabase**) e il blocco aperto del database (**OpenDatabase**).

Blocco per aprire il database SQLite (**OpenDatabase**)



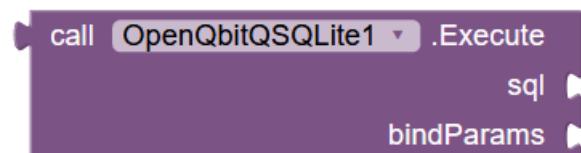
Unità obbligatoria: Block (**ImportDatabase**).

Parametri di ingresso: **Utilizzare prima di** < Dipendenza(i) obbligatoria(e) >

Parametri di output: Non applicabile, avviare o aprire un database SQLite per eseguire transazioni.

Descrizione: Blocco che avvia un database SQLite, ci deve essere prima.

Blocco per la query di dati in SQLite (**Execute**).



Unità obbligatorie: Block (**ImportDatabase**), Block (**OpenDatabase**).

Parametri di ingresso: **sql** < Stringa> , **bindParamelle** < Stringa> , **Utilizzare prima** < Dipendenza(e) obbligatoria(e) >.

Parametri di output: Lo statement SQL viene eseguito per creare una transazione in un database SQLite.

Descrizione: Il blocco che esegue gli statement SQL nel database SQLite, deve prima aver eseguito il blocco di importazione del database (**ImportDatabase**) e il blocco di apertura del database (**OpenDatabase**).

21. Definizione e uso dei blocchi di sicurezza.

In questa sessione esamineremo l'uso dei blocchi che ci forniscono livelli di sicurezza per salvare, convalidare e trasferire le transazioni dai nodi della rete Mini BlocklyChain.

I blocchi di sicurezza si basano sulla seguente estensione:

- I. Estensione OpenQbitAESEncryption.
- II. Estensione OpenQbitAESDecryption.
- III. Estensione OpenQbitAEToString.
- IV. Estensione OpenQbitEncDecData.
- V. Estensione OpenQbitFileHash.
- VI. Estensione OpenQbitRSA.
- VII. Estensione OpenQbitSSHClient (richiederebbe)
- VIII. Estensione OpenQbitStringHash.

Ad eccezione dell'estensione OpenQbitSSHClient che è obbligatoria, le estensioni di cui sopra sono opzionali da utilizzare nella creazione di una rete pubblica o privata Mini BlocklyChain.

Tuttavia, al fine di avere un sistema sicuro per le vostre transazioni in funzione di ogni business case, l'uso delle estensioni che sono opzionali dovrebbe essere applicato a vostra discrezione.

Ad esempio, nel caso dell'uso dell'hashish possiamo utilizzare una serie di opzioni di algoritmi come MD5, SHA1, SHA128, SHA256, SHA512 per una stringa di caratteri oltre ad essere applicato a qualsiasi tipo di file a seconda del flusso di informazioni di ogni sistema creato con Mini BlocklyChain.

Estensione OpenQbitAESEncryption.

Blocco per crittografare i file con sicurezza AES. (**AESEncryption**).



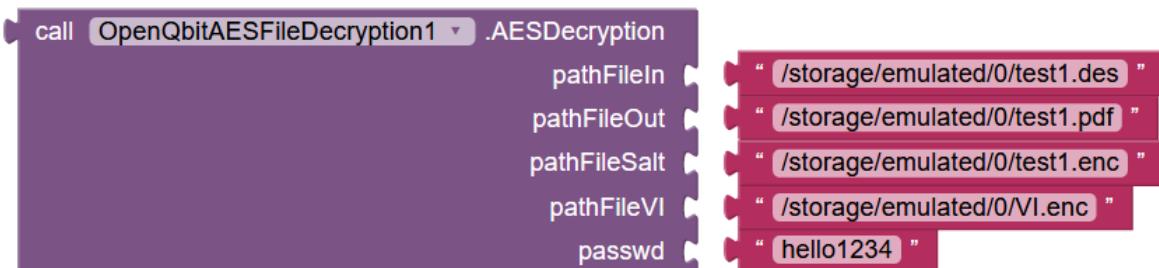
Parametri di ingresso: **pathFileIn < String>** , **pathFileOut < String>** , **pathFileFile < String>** , **pathFileVI < String>** e **passwd < String>**. I nomi dei file sono arbitrari e a discrezione di ogni progetto di sistema.

Parametri di uscita: file criptato AES inserito nel **percorso del** parametro di **ingressoFileIn**.

Descrizione: Blocco che ci dà tre file di output uno di questi è il file sorgente già criptato dall'algoritmo AES con una chiave a 256 bit, gli altri due file sono usati per controllare l'estensione per decriptare e recuperare il file originale.

Estensione OpenQbitAESDecryption.

Blocco per decrittare il file AES. (**AESDecryption**).



Parametri di ingresso: **pathFileIn < String>** , **pathFileOut < String>** , **pathFileFile < String>** , **pathFileVI < String>** e **passwd < String>**. I nomi dei file sono gli stessi di quelli ottenuti come risultato del blocco (**AESEncryption**).

Parametri di output: File originale decifrato con AES inserito nel parametro di input **pathFileIn**, in questo caso per decifrare il file viene inserito in **pathFileIn** il file criptato e in **pathFileOut** ci darà il file originale (decifrato).

Descrizione: Blocco che ci dà un file nel parametro **pathFileOut** che sarà l'uscita decifrata dall'algoritmo AES con una chiave a 256 bit.

Estensione OpenQbitAEToString.

Questa estensione è monouso per ogni sessione del dispositivo, cioè funziona solo quando il dispositivo non viene riavviato (telefono cellulare), poiché il valore di codifica VI viene generato temporaneamente.

NOTA: Se il dispositivo viene riavviato, la stringa criptata non può essere recuperata.

Blocco per la crittografia temporanea delle stringhe di caratteri (**DecrypSecretText**)

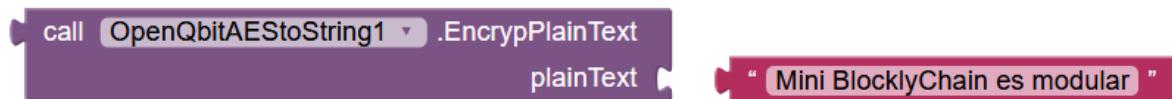


Parametri di ingresso: **secretText** < Stringa >

Parametri di uscita: stringa di caratteri originale decifrata con AES con chiave a 256 bit

Descrizione: Il blocco che ci dà una stringa di caratteri, è il parametro di input che è stato introdotto nel blocco (**EncrypPlainText**).

Blocco per decodificare una stringa (**EncrypPlainText**)



Parametri di ingresso: **plainText** < Stringa >

Parametri di uscita: stringa di caratteri criptati AES con chiave a 256 bit.

Descrizione: Blocco che ci fornisce una stringa di caratteri alfanumerici criptati con AES utilizzando una chiave digitale a 256 bit.

Estensione OpenQbitEncDecData.

Blocco di cifratura specializzato per database generici (**CrittografiaDati**)



Parametri di ingresso: **plainText** < Stringa >

Parametri di uscita: stringa di caratteri criptati AES con chiave a 256 bit.

Descrizione: Blocco che ci fornisce una stringa di caratteri alfanumerici criptati con AES utilizzando una chiave digitale a 256 bit.

Blocco di cifratura specializzato per database generici (**DescryptionData**)



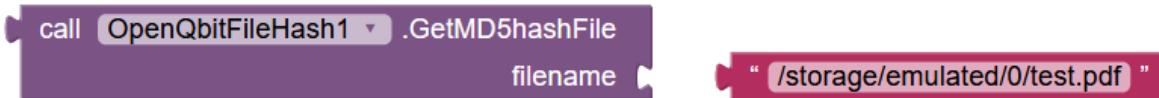
Parametri di ingresso: **plainText** <Stringa>

Parametri di uscita: stringa di caratteri criptati AES con chiave a 256 bit.

Descrizione: Blocco che ci fornisce una stringa di caratteri alfanumerici criptati con AES utilizzando una chiave digitale a 256 bit.

Estensione OpenQbitFileHash.

Blocco per generare MD5 da un file (**GetMD5hashFile**)

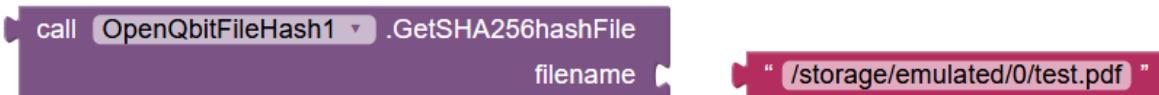


Parametri di ingresso: **nome del file** <stringa>

Parametri di uscita: Fornisce il file hash MD5.

Descrizione: Blocco per creare l'hash MD5 del file indicato nel parametro di input.

Blocco per generare SHA256 da un file (**GetSHA256hashFile**)

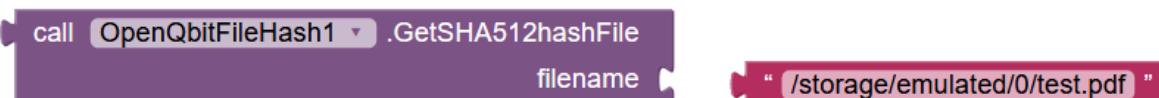


Parametri di ingresso: **nome del file** <stringa>

Parametri di uscita: Fornisce l'hash di archivio SHA256.

Descrizione: Blocco per creare l'hash SHA256 del file indicato nel parametro di input.

Blocco per generare SHA512 da un file (**GetSHA512hashFile**)

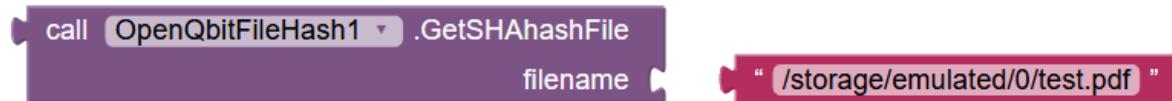


Parametri di ingresso: **nome del file** <stringa>

Parametri di uscita: Fornisce l'hash di archivio SHA256.

Descrizione: Blocco per creare l'hash SHA256 del file indicato nel parametro di input.

Blocco per generare SHA1 da un file (**GetSHA1hashFile**)



Parametri di ingresso: **nome del file** <stringa>

Parametri di uscita: Fornisce l'hash dell'archivio SHA1.

Descrizione: Bloccare per creare l'hash SHA1 del dado nel parametro di ingresso.

Estensione OpenQbitRSA.

Blocco per la **decrittazione delle** stringhe con RSA (**Decrypt**)



Dipendenze richieste: Blocco (**Encrypt**), Blocco (**OpenFromDiskPrivateKey**), Blocco (**OpenFromDiskPublicKey**).

Parametri di ingresso: **risultato** <Stringa>

Parametri di uscita: stringa di caratteri decodificata con RSA.

Descrizione: Blocco che ci dà una stringa di caratteri alfanumerici decifrati utilizzando la chiave della dimensione che è stata utilizzata nel blocco (**GenKeyValuePair**).

Blocco per la codifica della stringa con RSA (**Encrypt**)



Unità richiesta(e): Blocco (**GenKeyValuePair**), Blocco (**SaveFromDiskPrivateKey**), Blocco (**SaveFromDiskPublicKey**)

Parametri di ingresso: **semplice** <Stringa>

Parametri di uscita: stringa di caratteri criptati RSA

Descrizione: Blocco che ci dà una stringa di caratteri alfanumerici decifrati utilizzando la chiave della dimensione che è stata utilizzata nel blocco (**GenKeyValuePair**).

Blocco per la **decrittazione delle** stringhe con RSA (**Decrypt**)

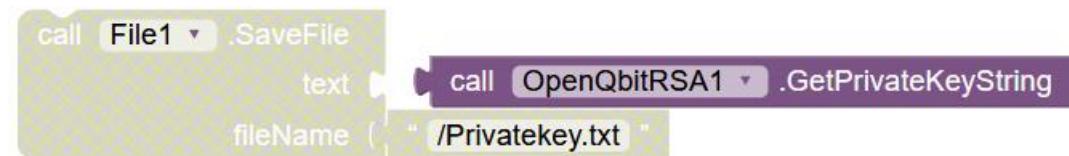


Parametri di ingresso: **dimensione <Intero**

Parametri di uscita: Non applicabile.

Descrizione: Blocco per generare la chiave privata e la chiave pubblica in base alla dimensione scelta.

Blocco per ottenere la chiave privata (**GetPrivateKeyString**)



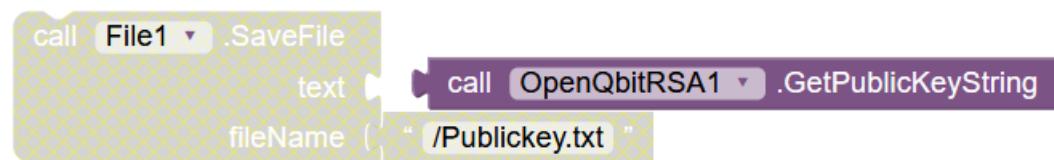
Unità obbligatoria: Block (**GenKeyValuePair**), Block (**GenKeyValuePair**), Block (**File**)

Parametri di ingresso: **Non applicabile.**

Parametri di uscita: File con stringa di caratteri criptati RSA (chiave privata)

Descrizione: Blocco che ci dà una stringa alfanumerica che rappresenta la chiave privata cifrata con la chiave di dimensione che è stata usata nel blocco (**GenKeyValuePair**).

Blocco per ottenere la chiave privata (**GetPublicKeyString**)



Unità obbligatoria: Block (**GenKeyValuePair**), Block (**GenKeyValuePair**), Block (**File**)

Parametri di ingresso: **Non applicabile.**

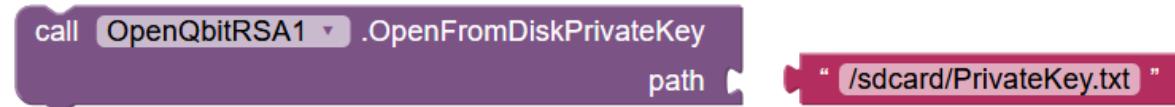
Parametri di uscita: File con stringa crittografata RSA (chiave pubblica)

Descrizione: Blocco che ci fornisce una stringa alfanumerica che rappresenta la chiave pubblica cifrata utilizzando la dimensione della chiave utilizzata nel blocco (**GenKeyPair**).

NOTA: Nei blocchi precedenti (**GetPrivateKeyString**) e (**GetPublicKeyString**) nelle dipendenze useremo il blocco generico (**File**) dell'applicazione App Inventor della sessione pallet "Storage".

Questo modo di conservare la chiave privata e pubblica tramite il blocco (**file**) può aiutarci ad avere una migliore manipolazione delle informazioni.

Bloccare per leggere la chiave privata da un file (**OpenFromDiskPrivateKey**).



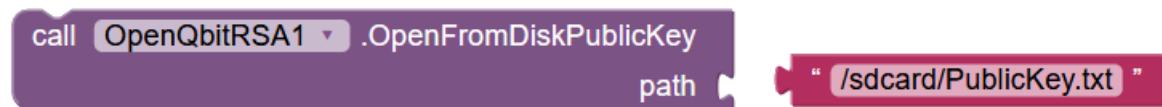
Dipendenze richieste: Block (**SaveFromDiskPrivateKey**) o Block (**GetPrivateKeyString**).

Parametri di ingresso: **percorso** < Stringa

Parametri di uscita: Caricamento del sistema chiave privata RSA stringa di caratteri criptati a chiave privata RSA.

Descrizione: Blocco che ci fornisce una stringa di caratteri alfanumerici criptati della chiave privata memorizzata nel percorso del file fornito.

Blocco per leggere la chiave pubblica da un file (**OpenFromDiskPublicKey**)



Unità obbligatoria: Block (**SaveFromDiskPublicKey**) o Block (**GetPublicKeyString**).

Parametri di ingresso: **percorso** < Stringa

Parametri di uscita: Caricare nel sistema la stringa di caratteri criptati a chiave pubblica RSA.

Descrizione: Blocco che ci fornisce una stringa alfanumerica criptata della chiave pubblica memorizzata nel percorso del file fornito.

Blocco per il salvataggio della chiave privata in un file (**SaveToDiskPrivateKey**).



Unità obbligatoria: Block (**GenKeyValuePair**).

Parametri di ingresso: **semplice** < Stringa

Parametri di uscita: File con stringa criptata RSA. (chiave privata)

Descrizione: Blocco che ci fornisce un file con una stringa alfanumerica criptata con chiave privata della dimensione utilizzata nel blocco (**GenKeyValuePair**).

Blocco per il salvataggio della chiave privata in un file (**SaveToDiskPublicKey**).



Unità obbligatoria: Block (**GenKeyValuePair**).

Parametri di ingresso: **semplice** < Stringa

Parametri di uscita: File con stringa criptata RSA. (chiave pubblica)

Descrizione: Blocco che ci dà un file con una stringa alfanumerica criptata usando una chiave pubblica della dimensione che è stata usata nel blocco (**GenKeyValuePair**).

Estensione OpenQbitSSHClient.

Blocco connettore SSH Client (**ConnectorMiniBlocklyChain**)



Parametri di input: **username** <string>, **password** <string>, **host** <string>, **port**<integer>, **porta**<integer>.

Parametri di uscita: Se la connessione con il server ssh del terminale Termux ha successo, ci dà un messaggio; "**Connect SSH**", se non ha successo, ci dà un messaggio **NULL**.

Descrizione: Blocco di comunicazione per collegare Mini BlocklyChain al terminale Termux, tramite il protocollo di comunicazione SSH (Secure Shell).

Blocco per l'esecuzione dei comandi nel terminale Linux Termux



(CommandLineMiniBlocklyChain).

Parametri di ingresso: **comando** <stringa>

Parametri di uscita: dati variabili, a seconda del comando o del programma eseguito.

Descrizione: Il blocco di esecuzione dei comandi nel terminale Termux è il prerequisito per effettuare una connessione con il blocco (**ConnectorMiniBlocklyChain**) può eseguire tutti i tipi di comandi online e/o ottenere dati di esecuzione specifici da script o programmi che hanno una CLI (Command-Line Interface) online.

ScollegareMiniBlocklyChainSSH.

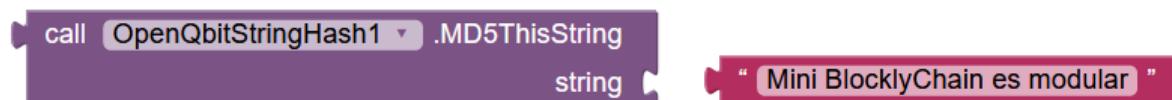


Parametri di ingresso e di uscita: Non applicabile (nessuno)

Descrizione: Bloccare per chiudere la sessione SSH.

Estensione OpenQbitStringHash.

Blocco per generare la stringa MD5 (**MD5ThisString**)

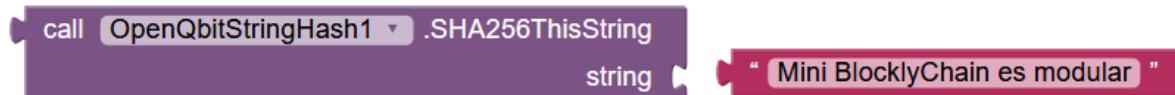


Parametri di ingresso: stringa

Parametri di uscita: Fornisce l'hash MD5.

Descrizione: Blocco per creare l'hash MD5 della stringa indicata nel parametro di ingresso.

Blocco per generare la stringa di caratteri SHA256 (**SHA256ThisString**)

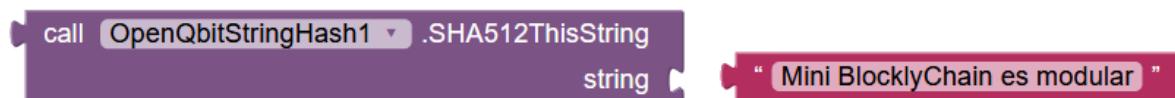


Parametri di ingresso: stringa

Parametri di uscita: Fornisce l'hash SHA256.

Descrizione: Blocco per creare l'hash SHA256 della stringa data nel parametro di ingresso.

Blocco per generare la stringa SHA512 (**SHA512ThisString**)

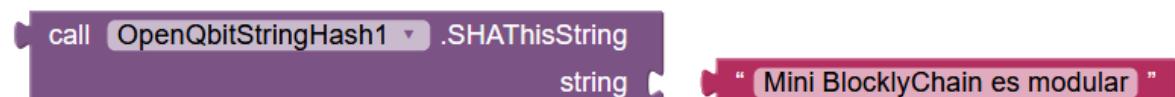


Parametri di ingresso: stringa

Parametri di uscita: Fornisce l'hash SHA512.

Descrizione: Blocco per creare l'hash SHA512 della stringa data nel parametro di ingresso.

Blocco per generare la stringa di caratteri SHA1 (**SHAThisString**)



Parametri di ingresso: stringa

Parametri di uscita: Fornisce l'hash SHA1.

Descrizione: Blocco per creare l'hash SHA1 della stringa indicata nel parametro di ingresso.

22. Impostazione dei parametri di sicurezza in Mini BlocklyChain.

I parametri di sicurezza sono suddivisi in tre componenti di qualsiasi sistema che viene progettato e applicato nei seguenti componenti:

- a. Redis database (rete di comunicazione di backup)
 - b. Sistema di sincronizzazione peer to peer
 - c. Integrare la sicurezza per proteggere il database SQLite
 - d. Ambiente di sviluppo per l'integrazione dei moduli
-
- a. Redis database (rete di comunicazione di backup)

Rinominando i comandi pericolosi, l'ulteriore funzione di sicurezza integrata di Redis comporta la ridenominazione o la disabilitazione di alcuni comandi considerati pericolosi.

Se eseguiti da utenti non autorizzati, questi comandi possono essere utilizzati per riconfigurare, distruggere o cancellare i loro dati. Come la password di autenticazione, la ridenominazione o la disabilitazione dei comandi è configurata nella stessa sezione SECURITY del file `/etc/redis/redis.conf`.

Alcuni dei comandi considerati pericolosi: **FLUSHDB**, **FLUSHALL**, **KEYS**, **PEXPIRE**, **DEL**, **CONFIG**, **SHUTDOWN**, **BGWRITEAOF**, **BGSAVE**, **SAVE**, **SPOP**, **SREM**, **RENAME** e **DEBUG**. Questa non è una lista completa, ma rinominare o disabilitare tutti i comandi di quella lista è un buon inizio per migliorare la sicurezza del vostro server Redis.

A seconda delle vostre esigenze specifiche o di quelle del vostro sito, dovete rinominare o disattivare un comando. Se sai che non userai mai un comando che può essere manipolato, puoi disabilitarlo. D'altra parte, potreste volerla rinominare.

Per abilitare o disabilitare i comandi di Redis, riaprire il file di configurazione:

```
$ vi /etc/redis/redis.conf
```

Attenzione: I seguenti passi per disabilitare e rinominare i comandi sono esempi. Dovreste scegliere di disattivare o rinominare solo i comandi che vi riguardano. È possibile rivedere l'elenco completo dei comandi e determinare come possono essere utilizzati in modo improprio in redis.io/comandi.

Per disabilitare un comando, è sufficiente rinominarlo in modo che diventi una stringa vuota (simboleggiata da una coppia di virgolette senza caratteri tra loro), come mostrato di seguito:

`/etc/redis/redis.conf`

```
    . . .
# È anche possibile uccidere completamente un comando rinominandolo in
# una stringa vuota:
#
rinominare il comando FLUSHDB "".
rinominare il comando FLUSHALL "".
rinomina-comando DEBUG "" ""
    . . .
```

Per rinominare un comando, assegnargli un altro nome come mostrato negli esempi seguenti. I comandi rinominati dovrebbero essere difficili da indovinare per gli altri, ma facili da ricordare per voi.

/etc/redis/redis.conf

```
    . . .
# rinomina comando CONFIG ""
rinominare il comando SHUTDOWN_MENOT
rinominare il comando CONFIG ASC12_CONFIG
    . . .
```

Salvare le modifiche e chiudere il file.

Dopo aver rinominato un comando, applicare la modifica riavviando Redis:

- sudo systemctl restart redis.service

Per testare il nuovo comando, inserire la riga di comando Redis:

- redis-cli

Poi, eseguire l'autenticazione:

- autentica la tua password_redis_password

Uscita
OK

Come nell'esempio precedente, supponiamo di rinominare il comando CONFIG in ASC12_CONFIG. Per prima cosa, provare ad usare il comando CONFIG originale. Perché l'hai ribattezzato, non dovrebbe funzionare:

- config get requirepass

Uscita
(errore) ERR comando sconosciuto 'config'.

Tuttavia, il comando rinominato può essere richiamato con successo. Non è un caso sensibile:

- asc12_config get requirepass

Uscita

- 1) "requirepass".
- 2) "your_redis_password" (la tua password)

Infine, potrete chiudere la riorganizzazione:

- uscita

Si noti che se si utilizza già la riga di comando Redis e si riavvia Redis, sarà necessario autenticarsi di nuovo. Altrimenti, se si digita un comando, questo errore verrà visualizzato:

Uscita
È richiesta l'autenticazione di NOAUTH.

b. Sistema di sincronizzazione peer to peer

Nell'utilizzo del sistema "Peer to Peer" tra i nodi il sistema ha i seguenti tre elementi applicati nella rete di comunicazione

-Privato: non ci sono informazioni memorizzate in nessun altro luogo che non siano i vostri computer. Non esiste un server centrale che possa essere compromesso (legalmente o illegalmente).

-Crittografato: tutte le comunicazioni sono protette attraverso il protocollo TLS (Transport Layer Security); un protocollo crittografico che include una sequenza perfetta per impedire a chiunque al di fuori della vostra sicurezza di accedere alle vostre informazioni.

-Autenticato: ogni nodo è identificato con un forte certificato crittografico. Solo i nodi che avete esplicitamente permesso, possono connettersi alle vostre informazioni.

<https://docs.syncthing.net/users/security.html>

Utilizzando il comando online SyncthingManager:

<https://github.com/classicsc/syncthingmanager>

c. Integrare la sicurezza per proteggere il database SQLite

Quando si usa l'estensione (**OpenQbitSQLite**) o nel suo caso l'estensione (**ConnectorSSHClient**) per usare la CLI SQLite entrambe le estensioni possono essere combinate con l'estensione di sicurezza AES (**OpenQbitEncDecData**).

Vedi appendice "Esempio di creazione di un sistema Mini BlocklyChain".

d. Ambiente di sviluppo per l'integrazione dei moduli

Per implementare la sicurezza nell'ambiente di sviluppo si può fare con due processi:

- Limitare l'uso delle librerie OpenJDK.
- L'uso è limitato ai nodi di sistema autorizzati.

23. Allegato "Creazione di banche dati KeyStore & PublicKeys".

Un KeyStore è un deposito di certificati di sicurezza, sia certificati di autorizzazione che certificati di chiave pubblica, password o chiavi di sicurezza generiche come le corrispondenti chiavi private (indirizzi) di un utente, che viene utilizzato per creare o elaborare transazioni.

Si tratta di un database criptato in modo che solo il proprietario possa utilizzarlo. Normalmente è di tipo locale, tuttavia, nel sistema Mini BloclyChain è un database sicuro ma viene condiviso e distribuito in tutti i nodi, questo è dovuto fondamentalmente ad una semplice ragione, tutti i nodi devono conoscere gli indirizzi di tutti gli utenti (indirizzi pubblici), gli indirizzi privati sono sempre locali e sono di uso unico ed esclusivo di ogni utente, tuttavia quando si effettua una transazione di entrata (deposito) o di uscita (spesa) ogni transazione ha sempre almeno tre componenti quando viene creata: indirizzo di origine, indirizzo di destinazione e bene o valore che viene inviato.

Per creare il nostro KeyStore dovremo seguire i seguenti passi e requisiti.

Un primo requisito è quello di avere un tipo di storage dove saranno memorizzate, nel nostro caso useremo il database SQLite e creeremo due KeyStore uno con le chiavi private dell'utente che sarà locale e sarà protetto "criptato" l'informazione e un altro database che memorizzerà le chiavi pubbliche questa base sarà condivisa in tutti i nodi della rete, la base che sarà condivisa nella rete sarà anch'essa criptata, tuttavia questa avrà una password che solo i membri (nodi) della rete potranno condividere.

Secondo requisito fondamentale è il processo di cifratura delle informazioni, questo sarà fatto con l'estensione specializzata per l'uso in database generico chiamato (**OpenQbitEncDecData**) che utilizza un algoritmo AES.

L'estensione (**OpenQbitEncDecData**) è funzionale alla verifica degli elementi unitari della catena a blocchi, tuttavia, nel caso di dover verificare la totale integrità della catena a blocchi non sarà efficiente quindi si effettuerà anche una cifratura di una copia, ma in questo caso sarà attraverso le estensioni: (**OpenQbitAESEncryption**) e (**OpenQbitAESDecryption**) che funzionano su un file, non con dati di riferimento.

NOTA: Il server SSH deve essere in funzione sul terminale Termux affinché il database di **KeyStore** funzioni correttamente. Eseguire il comando nel terminale:

```
$ sshd
```

Le estensioni basate sull'algoritmo AES possono essere utilizzate per qualsiasi tipo di dati o file e questo algoritmo è stato scelto in quanto è l'unico che si è dimostrato essere una prova contro gli attacchi basati sul calcolo quantistico.

Criptosistema	Categoría	Tamaño de clave	Parámetro de seguridad	Algoritmo cuántico estimado que rompa el criptosistema	Nº de qubits lógicos necesarios	Nº de qubits físicos necesarios	Tiempo necesario para romper el sistema	Estrategias de reemplazo cuántico-resilientes
AES-GCM	Cifrado simétrico	128	128	Algoritmo de Grover	2.953	$4,61 \times 10^6$	$2,61 \times 10^{12}$ años	
		192	192		4.449	$1,68 \times 10^7$	$1,97 \times 10^{22}$ años	
		256	256		6.681	$3,36 \times 10^7$	$2,29 \times 10^{32}$ años	
RSA	Cifrado asimétrico	1024	80	Algoritmo de Shor	2.290	$2,56 \times 10^6$	3,58 horas	Migrar a un algoritmo PQC seleccionado por el NIST
		2048	112		4.338	$6,2 \times 10^6$	28,63 horas	
		4096	128		8.434	$1,47 \times 10^7$	229 horas	
ECC Problema del logaritmo discreto	Cifrado asimétrico	256	128	Algoritmo de Shor	2.330	$3,21 \times 10^6$	10,5 horas	Migrar a un algoritmo PQC seleccionado por el NIST
		386	192		3.484	$5,01 \times 10^6$	37,67 horas	
		512	256		4.719	$7,81 \times 10^6$	95 horas	
SHA256	Minado de Bitcoin	N/A	72	Algoritmo de Grover	2.403	$2,23 \times 10^6$	$1,8 \times 10^6$ años	
PBKDF2 con 10.000 iteraciones	Hashing de contraseñas	N/A	66	Algoritmo de Grover	2.403	$2,23 \times 10^6$	$2,3 \times 10^7$ años	Abandonar la autenticación basada en contraseñas

La tabella sopra riportata è riferita alle Accademie Nazionali di Scienza, Ingegneria e Medicina.

<https://www.nap.edu/catalog/25196/quantum-computing-progress-and-prospects>

Descrizione

La meccanica quantistica, il sottocampo della fisica che descrive il comportamento delle particelle molto piccole (quantum), fornisce la base per un nuovo paradigma di calcolo. Proposto per la prima volta negli anni '80 come modo per migliorare la modellazione computazionale dei sistemi quantistici, il campo del calcolo quantistico ha recentemente attirato una notevole attenzione grazie ai progressi nella costruzione di dispositivi su piccola scala. Tuttavia, saranno necessari notevoli progressi tecnici prima che un computer quantistico pratico possa essere realizzato su larga scala.

Quantum Computing: Progress and Prospects fornisce un'introduzione al settore, comprese le caratteristiche uniche e i limiti della tecnologia, e valuta la fattibilità e le implicazioni della creazione di un computer quantistico funzionale in grado di affrontare i problemi del mondo reale. Questo rapporto prende in considerazione i requisiti hardware e software, gli algoritmi quantistici, i driver dei progressi nel calcolo quantistico e nei dispositivi quantistici, i benchmark associati ai casi d'uso rilevanti, il tempo e le risorse necessarie e come valutare la probabilità di successo.

Installazione del gestore di database SQLite nel terminale TERMUX e testare il CLI (Command-Line) del comando **sqlite3** creando un database chiamato **keystore.db**

Noi usiamo i comandi:

\$ apt install sqlite

\$ sqlite3 keystore.db

```
$ apt install sqlite
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  sqlite
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 459 kB of archives.
After this operation, 811 kB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm sqlite arm 3.32.2-3 [459
kB]
Fetched 459 kB in 1s (237 kB/s)
Selecting previously unselected package sqlite.
(Reading database ... 16937 files and directo
ries currently installed.)
Preparing to unpack .../sqlite_3.32.2-3_arm.deb
...
Unpacking sqlite (3.32.2-3) ...
Setting up sqlite (3.32.2-3) ...
$
```

```
$ sqlite3 keystore.db
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
sqlite> .databases
main: /data/data/com.termux/files/home/mcb/keyst
ore.db
sqlite> .quit
$ ls
keystore.db
$
```

Poi all'interno del **gestore sqlite>** eseguiamo la frase `.databases` per verificare in quale percorso è il database che stiamo creando, poi per uscire e salvare il database diamo la frase di `. quit`.

NOTA: In entrambe le affermazioni o comandi all'interno **del gestore sqlite>** bisogna prima mettere un punto `".`" **nella** sintassi.

Infine controlliamo che il database (vuoto) sia già stato creato dando il comando:

\$ ls

Si procede alla creazione di una tabella in cui le chiavi primarie saranno memorizzate in tre diversi formati: esadecimale, binario e l'indirizzo di riferimento utente Mini BlocklyChain che è la chiave pubblica della rispettiva chiave primaria.

La crittografia dei dati AES verrà applicata solo in formato esadecimale e binario. Nel caso dell'indirizzo dell'utente addrMBC e dell'alias non perché sia la chiave pubblica che può e deve essere condivisa in rete per ricevere le transazioni, così come l'alias per effettuare la ricerca da questo campo.

Ha creato una tabella chiamata "privatekey" nel database in SQLite (keystore.db).

```
TAVOLA CREARE TAVOLA Privata (
    id tasto primario intero
    alias      VARCHAR(50) NON NULLO
    addrHexVARCHAR  (65) NOT NULL
    addrMBCVARCHAR  (65) NON NULLA
    addrBinBLOB  NON NULLA
);
```

Eseguiamo le frasi nel CLI sqlite per creare il database di keystore.db.

Usiamo di nuovo la riga di comando sqlite3 con il seguente comando:

\$ sqlite3

Questo ci manderà all'interno del gestore del **database sqlite> database** manager. In questo primo apriremo il database già creato per poterci lavorare con la seguente frase all'interno del gestore:

Sqlite> .open keystore.db

Poi inseriremo lo statement SQL "**CREATE TABLE**" per creare la tabella della **chiave privata**.

Successivamente, vengono mostrate due opzioni per creare la stessa tabella, una è attraverso una singola linea dove sono inclusi tutti gli statement SQL degli elementi che la integrano. Il secondo esempio è attraverso l'introduzione di ogni elemento che integra la struttura in modo segmentato.

\$ sqlite3

SQLite versione 3.32.2 2020-06-20 15:25:24

Inserire ".help" per i suggerimenti d'uso.

Collegato ad un database di memoria transitoria.

Utilizzare ".open FILENAME" per riaprire un database persistente.

sqlite> .open keystore.db

sqlite> CREATE TABLE privatekey (id integer chiave primaria AUTOINCREMENT NOT NULL, alias VARCHAR(50) NOT NULL, addrHex VARCHAR(65) NOT NULL, addrMBC VARCHAR(65) NOT NULL, addrBin BLOB NOT NULL);

sqlite> .quit

La creazione della stessa tabella delle **chiavi private** viene mostrata in seguito, ma introducendo lo statement SQL in modo segmentato:

sqlite> CREARE TAVOLO Privato (

...> id tasto primario intero AUTOINCREMENTO
...> alias VARCHAR(50) NOT NULL,
...> addrHex VARCHAR (65) NOT NULL,
...> addrMBC VARCHAR (65) NOT NULL,
...> addrBin BLOB NON NULLA
...>);

sqlite> .tavoli

Privatekey

sqlite> .quit

I due esempi sopra riportati danno lo stesso risultato. In seguito eseguiamo la frase **.tables** per verificare che la tabella delle chiavi private sia stata creata, alla fine daremo la frase **.quit** con questo processo abbiamo già creato la tabella delle **chiavi private** all'interno del database SQLite **keystore.db**.

Fino a questo momento abbiamo già la struttura del database di **keystore.db** e la sua tabella delle chiavi private dove le chiavi private saranno memorizzate in modo criptato.

Questo dovrebbe mostrare qualcosa di simile nel nodo (telefono cellulare) con il terminale TERMUX.



```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open keystore.db
sqlite> CREATE TABLE privatekey (
...>     id integer primary key AUTOINCREMENT,
...>     alias VARCHAR(50) NOT NULL,
...>     addrHex VARCHAR(65) NOT NULL,
...>     addrMBC VARCHAR(65) NOT NULL,
...>     addrBin BLOB NOT NULL
...> );
sqlite> .tables
privatekey
sqlite> .quit
$
```

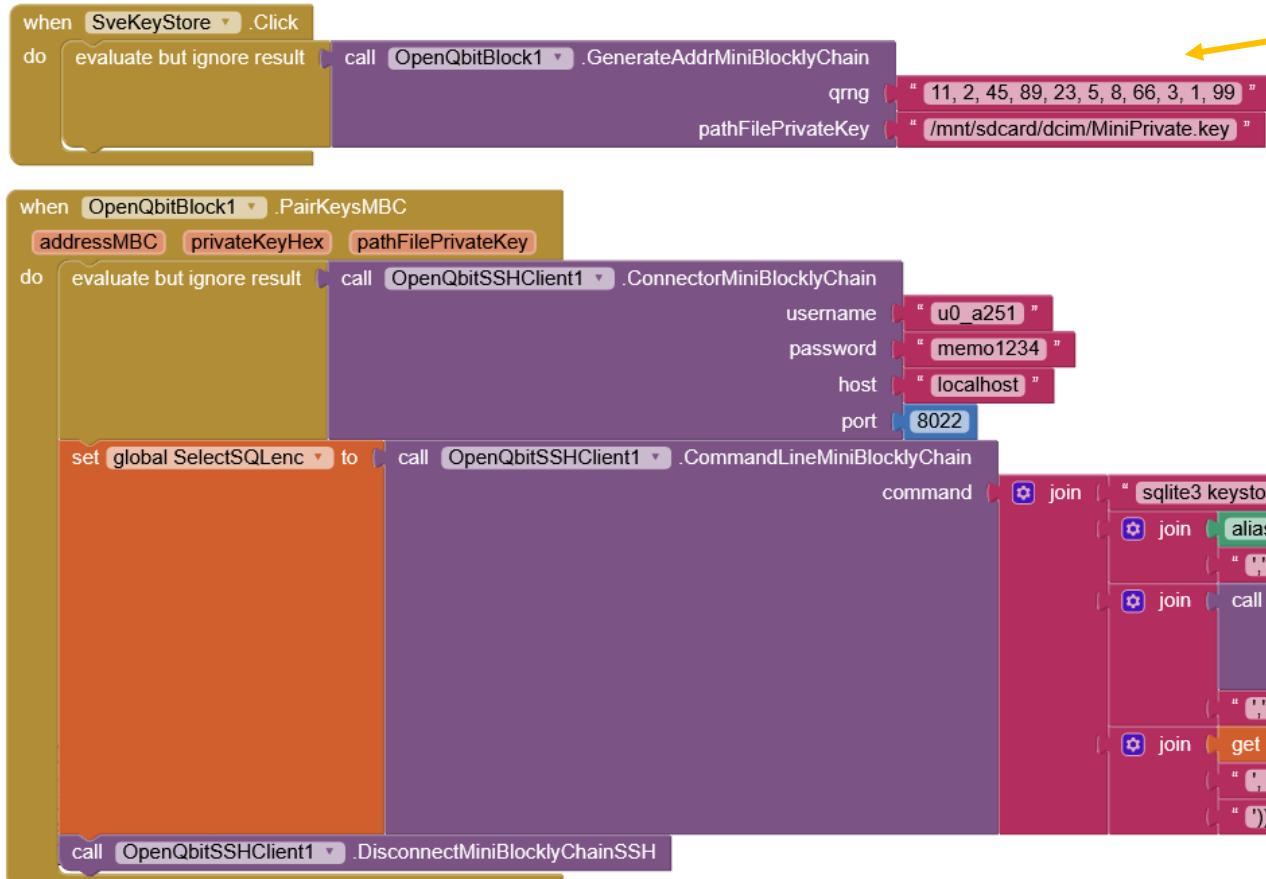
Dichiarazioni SQL
per creare nuove
tabelle

Utilizzeremo ora alcune estensioni di sicurezza per inserire i dati della "chiave privata" e quindi consultare questi dati.

Utilizzeremo il blocco per generare un nuovo indirizzo utente (**GenerateAddrMiniBlocklyChain**), questo blocco ci darà l'indirizzo privato in due formati (esadecimale e binario), così come l'indirizzo pubblico in formato indirizzo generico MBC. Utilizzeremo anche l'estensione (**OpenQbitSSHClient**) e l'estensione (**OpenQbitEncDecData**)

per vedere maggiori dettagli di questi controlli nella sezione "Definizione e uso dei blocchi di sicurezza". Nel terminale Termux dovrebbe essere in funzione il servizio SSH.

INSERIRE dati criptati nel database di **keystore.db**

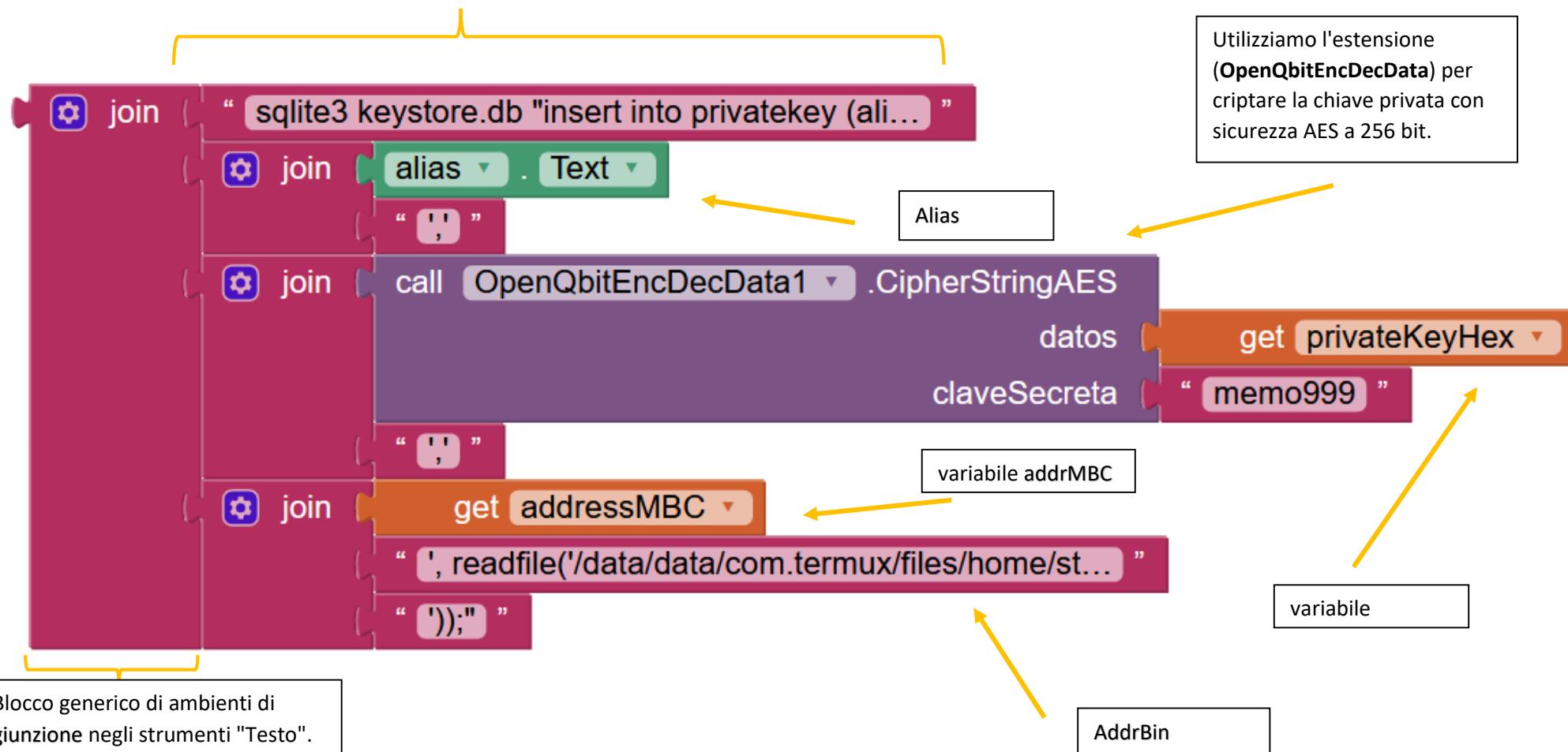


Serie di numeri casuali quantistici generati dal blocco (ApiGetQRNGInteger) verificare come formattare il risultato JSON, poiché l'immissione del blocco (GenerateAddrMiniBlocklyChain) richiede una serie di numeri separati solo da virgole ",".

La sintassi del comando è molto importante, dobbiamo introdurre il seguente comando nel giusto formato nell'ambiente Blockly.

I valori dei valori saranno sempre le variabili, ad esempio:

```
sqlite3 keystore.db "inserire in privatekey (alias, addrHex, addrMBC, addrBin) valori ('memo', 'QWERTY', 'MBC12345',  
readfile('/data/data/com.termux/files/home/storage/dcim/MiniPrivate.key');");".
```



Dopo l'esecuzione dei blocchi avremo un risultato nella tabella delle chiavi private del database di keystore.db simile al seguente:

Entriamo nel gestore sqlite del terminale Termux, apriamo la base di keystore.db ed eseguiamo la frase:

\$ sqlite3

SQLite version 3.32.2 2020-06-20 15:25:24

Inserire ".help" per i suggerimenti d'uso.

Collegato ad un database di memoria transitoria.

Utilizzare ".open FILENAME" per riaprire un database persistente.

sqlite> .open keystore.db

sqlite> selezionare * da privatekey;

```

$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open keystore.db
sqlite> select * from privatekey;
1|mexico|07JBBizTcC0Ce6a8Pwe5aTV41Ql1DKiUQZyPSfJuRbVUZnvIwQJcry4LilBMs5jHavQeMo1iN8rCO87V5Xka2YaKR4fswopeTQmI/Q+ipzI=|2eENpFt2HSuGtXNxoeiwfrp5d6e87Z7y5|0♦♦
sqlite>

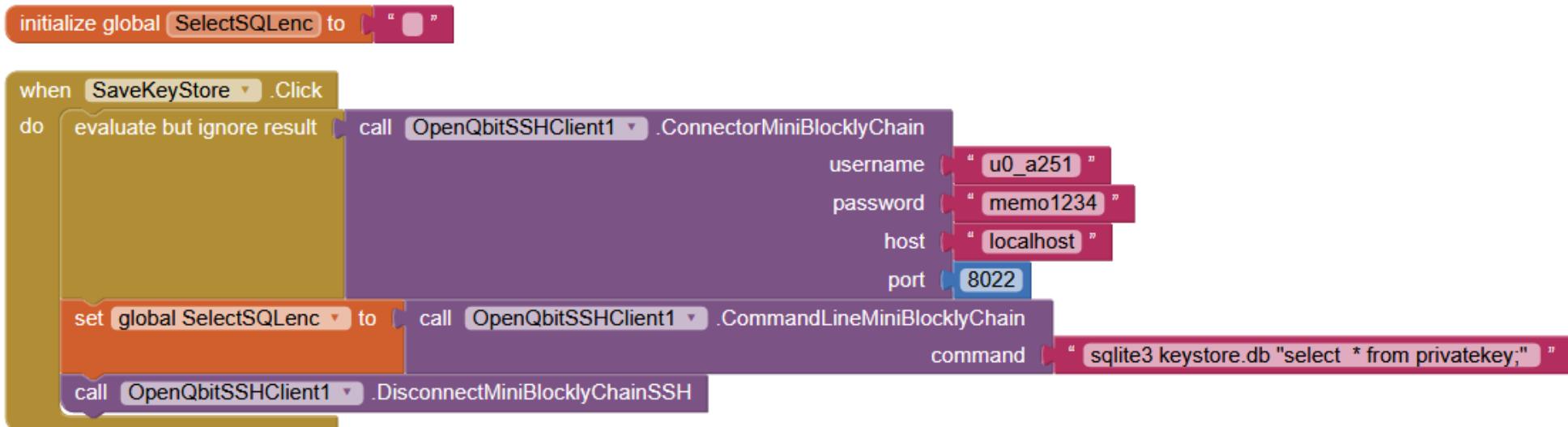
```

Alias: messico

addrBin (chiave privata in

cifratura

CONSULTARE tutti i dati nella tabella delle **chiavi private** del database SQLite **keystore.db**



CONSULTARE gli alias dei dati nella tabella delle **chiavi private** del database SQLite **keystore.db**

sqlite3 keystore.db "seleziona alias da privatekey;"

Interroga tutti i dati della colonna addrHex criptata nella tabella delle **chiavi private** del database SQLite **keystore.db**

sqlite3 keystore.db "seleziona addrHex da privatekey;".

CONSULTARE per recuperare la chiave privata addrBin nella tabella delle **chiavi private** del database SQLite **keystore.db**

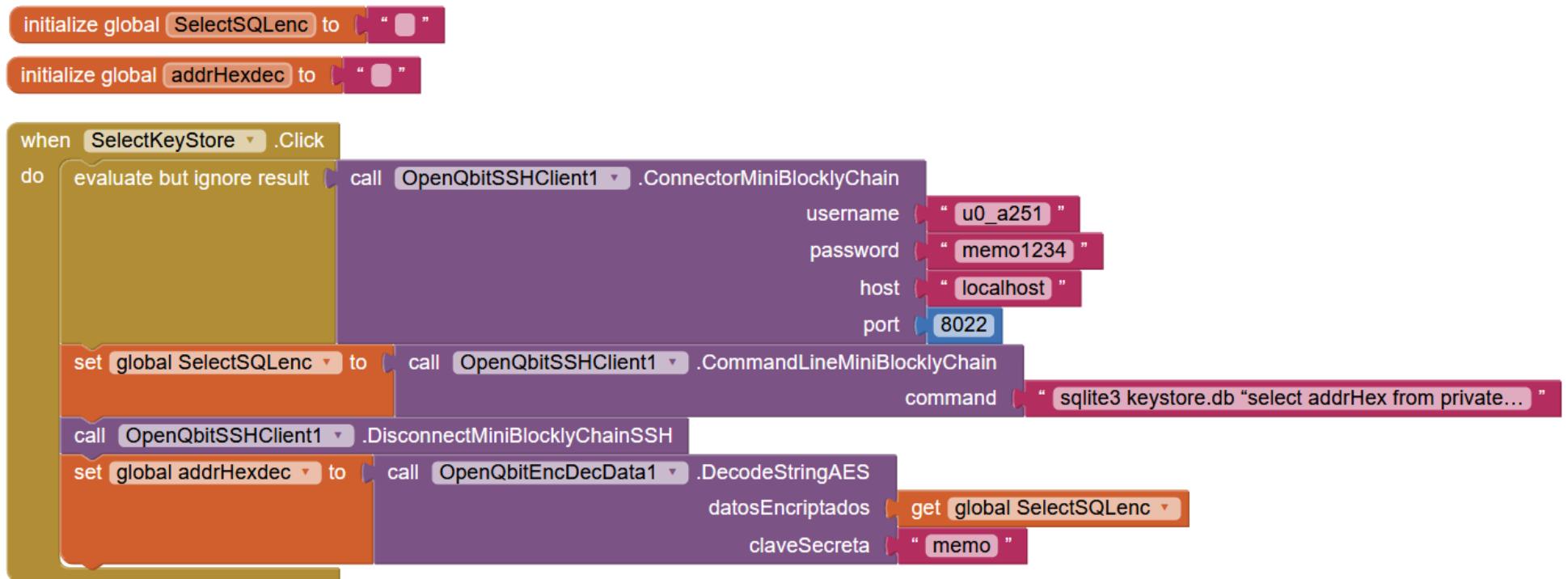
sqlite3 **writefile('PrivateKey.key', addrBin)** DA privatekey WHERE alias='mexico'; (2)

(1) Questo tipo di interrogazione sarà il principale per consultare la chiave privata per eseguire il processo di firma delle transazioni.

Query per la decodifica dei dati da parte di Alias nella colonna addrHex della tabella delle **chiavi private del database SQLite keystore.db**

In questo caso useremo il blocco (**DecodeStringAES**) per decodificare i dati memorizzati nella colonna "addrHex".

sqlite3 keystore.db "seleziona addrHex da privateKey where='mexico';".



Questa consultazione ci fornisce la chiave privata in formato esadecimale, questa è la parte fondamentale di qualsiasi ricezione o invio di transazioni. Si raccomanda ripetutamente di conservare un backup di questo database keystore.db.

Database design **publickeys.db** con tabella publicaddr:

```
$ sqlite3
```

SQLite version 3.32.2 2020-06-20 15:25:24

Inserire ".help" per i suggerimenti d'uso.

Collegato ad un database di memoria transitoria.

Utilizzare ".open FILENAME" per riaprire un database persistente.

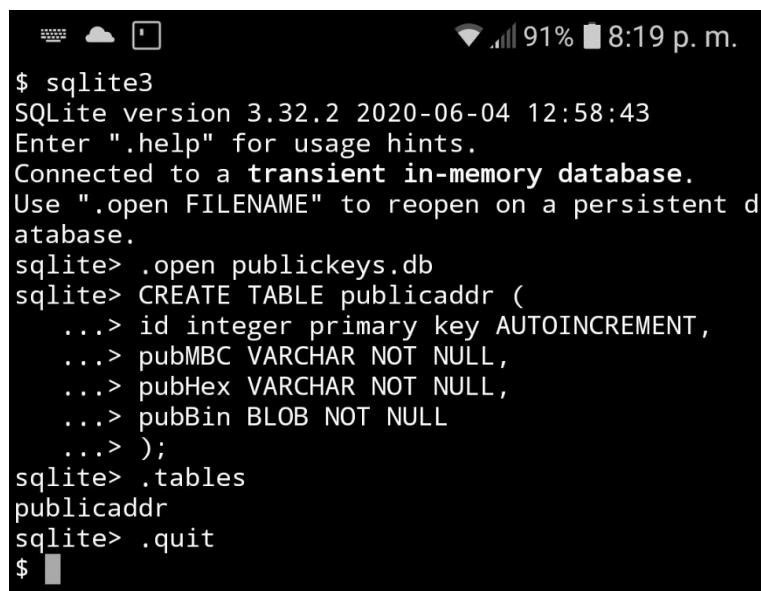
```
sqlite> .open publickeys.db
```

```
sqlite> CREATE TABLE publicaddr (id integer chiave primaria AUTOINCREMENT NOT NULL,  
pubMBC VARCHAR NOT NULL, pubHex VARCHAR NOT NULL, pubBin BLOB NOT NULL);
```

```
sqlite> .quit
```

Successivamente, la creazione della tabella **pubblicata** viene mostrata con la seguente frase SQL in forma segmentata:

```
sqlite> TAVOLA CREARE pubblicatoaddr (  
...> id tasto primario intero AUTOINCREMENTO  
...> pubMBC VARCHAR NON NULLA,  
...> pubHex VARCHAR NON NULLA,  
...> pubBin BLOB NON NULLA  
...> );  
sqlite> .tavoli  
publishedaddr  
sqlite> .quit
```



The screenshot shows a terminal window on a mobile device. The status bar at the top indicates signal strength, battery level (91%), and the time (8:19 p.m.). The terminal prompt is '\$'. The user runs 'sqlite3' to open a database. SQLite version 3.32.2 from June 2020 is loaded. The user then creates a table named 'publicaddr' with four columns: 'id' (primary key, auto-increment), 'pubMBC' (VARCHAR, not null), 'pubHex' (VARCHAR, not null), and 'pubBin' (BLOB, not null). Finally, the user lists tables with '.tables' and exits with '.quit'.

```
$ sqlite3  
SQLite version 3.32.2 2020-06-04 12:58:43  
Enter ".help" for usage hints.  
Connected to a transient in-memory database.  
Use ".open FILENAME" to reopen on a persistent database.  
sqlite> .open publickeys.db  
sqlite> CREATE TABLE publicaddr (  
...> id integer primary key AUTOINCREMENT,  
...> pubMBC VARCHAR NOT NULL,  
...> pubHex VARCHAR NOT NULL,  
...> pubBin BLOB NOT NULL  
...> );  
sqlite> .tables  
publicaddr  
sqlite> .quit  
$
```

24. Allegato "Comandi RESTful SQLite GET/POST".

Successivamente vengono mostrati i diversi comandi che possiamo utilizzare nel Restful SQLite della rete di backup. Questi sono stati consultati a partire dallo sviluppo originale di GITHUB (<https://github.com/olsonpm/sqlite-to-rest>)

Operazioni CRUD (Create, Read, Update and Delete) RESTful.

Di seguito è riportato un elenco delle operazioni disponibili messe a disposizione dalla RESTful API sotto forma di pseudo esempi. Assumeremo una base "op.sqlite" e due tabelle "trans" associate per le transazioni e un altro "segno" per l'indirizzo di origine, indirizzo di destinazione e valore del bene con due colonne id INTEGER PRIMARY KEY.

Come indicato nelle limitazioni, si noti che i metodi (CANCELLA e POST) possono interessare solo una riga alla volta.

OBTAIN Questo permette la più grande variazione. Più tardi vedremo tutti gli operatori di query disponibili.

Le intestazioni possono essere specificate appena sotto gli URL.

/transRichieste
per tutte le righe

/trans?id=1Dove
id = 1

/trans
gamma: righe=0-2Prime
tre righe

/trans
gamma: righe=-5Last
cinque righe

/trans
gamma: righe=0-

Tante righe quante il server può fornire, che in pratica sarà il minore tra maxRange e numero totale di righe.

/trans
gamma: righe=1-

Tante righe quante il server può fornire, a partire dalla riga 1.

/trans
ordine: nomeOrdinato
per nome ascendente

/trans
ordine: nome descOrdinato
per nome discendente

/trans
ordine: nome desc, id

Contributi, ma prima ordinati per nome discendente, e in caso di parità per id ascendente.

/trans?id>1
Dove id > 1

/trans?id>=2&id<5
Dove id >= 2 e id < 5

/trans?nome_NOTNULL
Dove il nome non è nullo

/trans?name_ISNULLDove il nome
è nullo

/trans?id!=5&nome_LIKE'Spotted%'.
Dove id != 5 e il nome è "Spotted%" (ignorare le virgolette)

/trans?id>=1&id<10&nome_LIKE'Avery%'.
ordine: nome desc, id range: rows=2-4

Ha contribuito a dare l'esempio.

Ottenere una transazione con identificatori compresi tra 1 e 9, con un nome come "Avery%", ordinato prima per nome discendente e poi per identificatore ascendente, ottenendo la terza e la quinta riga del risultato. O in SQL:

DELETE Richiede una stringa di interrogazione con tutte le chiavi primarie impostate pari a un valore. Ciò richiede una cancellazione massima di una sola riga.

/trans?id=1Deletes
trans con id=1

Se la transazione avesse invece una chiave principale composta da id e nome.

/trans?id=1&nome='Avery IPA'.

POST creare

Non è necessario passare una stringa di interrogazione. Se viene passata una stringa di interrogazione, si assume un aggiornamento POST. Tutte le richieste POST devono passare il tipo di contenuto dell'intestazione: application / json.

Si prega di notare che il corpo deve contenere tutte le colonne PRIMARY KEY non annullabili e non INTEGRATE. In caso contrario, verrà inviata una risposta di 400, indicando quali campi sono andati perduti. Le colonne nullabili saranno nulle e le colonne INTEGER PRIMARY KEY saranno automaticamente aumentate secondo le specifiche sqlite3.

I dati JSON saranno specificati appena sotto gli URL.

/trans

```
{"id":1,"name": "Serendipity"}Crea  
un trans con id = 1 e name = 'Serendipity'.
```

/trans

```
{"id":1}Crea  
un trans con id = 1 e nome = NULL
```

/trans

```
{"nome": "Serendipity"}
```

Crea una transazione con id impostato al seguente valore aumentato di sqlite3 Specifiche INTEGER PRIMARY KEY.

/trans

```
{ }
```

Crea una transazione con id aumentato e il nome impostato su NULL.

Aggiornamento POST

Deve contenere una stringa di interrogazione. Senza una stringa di interrogazione, si presume la creazione del POST. Come per POST create, è richiesto il tipo di contenuto dell'intestazione: application / json.

La stringa di interrogazione deve contenere tutte le chiavi primarie per garantire l'aggiornamento di una sola riga. Se vengono trasmessi valori errati, verrà restituito un 400 con i tasti offensivi.

Il corpo della richiesta deve contenere un oggetto non vuoto e deve contenere chiavi valide corrispondenti ai nomi delle colonne.

I dati JSON saranno specificati appena sotto gli URL.

/trans?id=1

```
{"id":2}
```

Aggiornare la transazione con un ID di 1 impostandola su due.

/trans?id=1

```
{ "nome" : " MCBza45Rt56cvbgfdR2Swd788kj" }
```

Aggiornare la transazione con l'ID di 1 impostando il suo nome o valore su "MCBza45Rt56cvbgfdR2Swd788kj".

Se il trading desk avesse invece una chiave principale composta da id e nome.

/trans?id=1&name=MCBza45Rt56cvbgfdR2Swd788kj

```
{ "nome" : " MCB3ofFG5Hj678MNb09vLdfaasX " }
```

Aggiornare la transazione dove l'id è uno e l'indirizzo è MCBza45Rt56cvbgfdR2Swd788kj, impostando il MCB3ofFG5Hj678MNb09vLdfaasX.

Riferimento

isSqliteFile

Basta controllare i primi 16 byte del file per vedere se è uguale al 'formato sqlite 3' seguito da un byte nullo.

isDirectory

Restituisce il risultato di fs.statsSync seguito da .isDirectory

isFile

Restituisce il risultato di fs.statsSync seguito da .isFile

Operatori di consulenza GET

Le condizioni di richiesta devono essere contrassegnate con simboli di collegamento, ad es. id> 5 & name = MCBza45Rt56cvbgfdR2Swd788kj

Operatori binari (richiede un valore dopo) Uomo.

=

!=

>=

<=

>

<

LICHIKE

LIKE è speciale perché deve avere quotazioni di apertura e chiusura semplici. In caso contrario, verrà generato un errore di 400 che indicherà dove non è stato possibile completare l'analisi e cosa ci si aspettava. Vedere CRUD RESTful Operations per esempi.

Singoli operatori (devono seguire il nome di una colonna)

È NULLA

NON NULLA

Oggetto configurazione del router

isLadenPlainObject

Lo scopo di questo oggetto è quello di fornire una configurazione generica per il router sqlite. Sono ammesse le seguenti proprietà:

prefisso: isLadenString La stringa passata all'opzione builder del prefisso del koa-router. Ad esempio, lo skeleton server non specifica un prefisso, che permette di colpire l'API della birra direttamente dalla radice del dominio `http://localhost: 8085 / trans`. Se si imposta il prefisso su '`/ api`', allora si devono inviare le richieste a `http://localhost: 8085 / api / trans`.

allTablesAndViews: un oggetto di configurazione tabulare

Le impostazioni specificate in questo oggetto saranno applicate a tutte le tabelle e le viste, eventualmente sovrascritte dalla proprietà `tablesAndViews`.

tablesAndViews: isLadenPlainObject L'oggetto passato deve avere le chiavi che corrispondono alla colonna del database o visualizzare i nomi. In caso contrario, verrà emesso un messaggio di errore amichevole. I valori per ogni tabella e vista devono essere un oggetto di configurazione tabulare.

Oggetto di configurazione tabellare

isLadenPlainObject Questo oggetto rappresenta le impostazioni che possono essere impostate per le viste o le tabelle. Permette le seguenti proprietà:

maxRange: isPositiveNumber

Applicazione predefinita: 1000

Questa è la portata massima consentita dal vostro server. Se una richiesta GET arriva senza un'intestazione del range, la specifica presuppone che si voglia l'intera risorsa. Se il numero di righe che risulta in GET è maggiore di `maxRange`, viene restituito uno stato 416 con l'intestazione `max-range` personalizzata. Il valore di default dell'applicazione è volutamente conservativo nella speranza che gli autori impostino `maxRange` in base alle loro esigenze.

Si prega di notare che 'Infinito' è un numero positivo valido.

bandiere: isLadenArray

Attualmente, l'unico indicatore accettato è la stringa '`sendContentRangeInHEAD`'. Una volta impostate, le richieste HEAD restituiranno la gamma di contenuti disponibili nel modulo

content-range: * / <max-range>. La ragione per cui è configurabile è che il calcolo del range massimo può essere più impegnativo di quanto valga, a seconda del carico del server e della dimensione delle vostre tabelle.

Testate personalizzate

Applicazione

ordine: questa intestazione è definita solo per GET, e può essere considerata come l'equivalente di sql ORDER BY. Deve contenere un nome di colonna delimitato da virgole, seguito da uno spazio e dalle stringhe 'asc' o 'desc'. Se vengono inviati valori d'ordine errati, una risposta di 400 indicherà quali.

Rispondi a

Non tutti sono necessariamente personalizzati, ma tutti gli utilizzi sono al di fuori delle specifiche e quindi necessitano di chiarimenti.

GET

max-range: questa intestazione viene restituita quando il numero di righe richieste supera il maxRange configurato. Si noti che la richiesta potrebbe non specificare l'intestazione dell'intervallo, ma il numero di righe risultanti in quella risorsa sarà comunque verificato.

gamma di contenuti: rfc7233 stati

Solo i codici di stato 206 (Contenuto parziale) e 416 (Intervallo insoddisfacente) descrivono un significato per Content-Range.

Quando sqlite-to-rest risponde con un codice di stato 200, l'intestazione dell'intervallo di contenuto viene inviata con il formato 206 di <filo inizio> - <filo fine> / <filo conteggio>.

Quando una richiesta viene inviata senza intestazione dell'intervallo e il numero di righe risultante supera il maxRange, viene restituito un 400 con l'intervallo di contenuto impostato nel formato 416 di * / <conteggio delle righe>.

Si noti che questa intestazione può essere restituita in una risposta HEAD.

accept-order: verrà restituito se l'ordine dell'intestazione della richiesta ha una sintassi errata o nomi di colonna non corretti. Per maggiori dettagli, vedere HEAD -> accetta-ordine qui sotto.

25. Allegato "Connettore Java Code SQLite-Redis Connector".

Di seguito viene mostrato un codice del collegamento tra la comunicazione di entrambi i database SQLite-Redis. Fondamentalmente, come possiamo vedere, il connettore cambia il formato SQLite in una stringa generica dei dati (transazioni) che Redis deve ricevere.

Il processo di cui sopra agisce come un trigger della coda di transazione a tutti i nodi che sono collegati e sono possibili candidati per l'elaborazione della coda di transazione corrente.

Quando il database Redis riceve la coda delle transazioni dalla configurazione Master-Slave di cui dispone, distribuirà le informazioni in tempo reale e in modo uguale a tutti i nodi.

Un altro punto fondamentale è che tutti i nodi devono essere sincronizzati nel loro orologio, questo sarà fatto come abbiamo visto prima con la funzionalità della query ad un pool di server NTP (Network Time Protocol).

Questo connettore esegue anche il primo livello di revisione della sicurezza dei dati calcolando l'albero Merkle Root di tutte le transazioni.

Codice base del connettore SQLite-Redis.

```
importare java.sql.*;
importare java.util.*;
importare java.util.array;
importare java.util.list;
importare java.util.array;
importare java.security.*;
importare java.security.MessageDigest;
importare redis.clients.jedis.Jedis;
classe RediSqlite{
    pubblico Stringa precedenteHash;
    pubblico String merkleRoot;
    public static ArrayList<String> transactions = nuova ArrayList<String>();
    public static ArrayList<String> treeLayer = nuova ArrayList<String>();
    pubblico statico String getMerkleRoot() {
        int count = transazioni.size();
        voce int = 1;
        Int dispari = 0;
        ArrayList<Stringa> previousTreeLayer = transazioni;
        mentre(conteggio > 1) {
            treeLayer = nuova ArrayList<Stringa>();
            per(int i=1; i <= contare; i=i+2) {

                se (!esPar(count) && i == count) dispari = 1;
                treeLayer.add(applySha256(previousTreeLayer.get(i-item)
                previousTreeLayer.get(i-impar)));
            }
        }
    }
}
```

+

```

voce = 1;
Dispari = 0;
count = treeLayer.size();
previousTreeLayer = treeLayer;
}

Stringa merkleRoot = (treeLayer.size() == 1) ? treeLayer.get(0) : "";
    restituire merkleRoot;
}

//Definire se Merkle Tree è pari o dispari
booleano statico enPar(int number){
    se (numero%2==0) restituisce vero; altrimenti restituisce falso;
}

public static String applySha256(String input){
    provare {
        MessageDigest digest = MessageDigest.getInstance ("SHA-256");
        //Applica sha256 al nostro input,
        byte[] hash = digest.digest(input.getBytes("UTF-8"));
        StringBuffer hexString = nuova StringBuffer(); // Questo conterrà hash come
esidecimale
        per (int i = 0; i < hash.length; i++) {
            Stringa hex = Integer.toHexString(0xff & hash[i]);
            if(hex.length() == 1) hexString.append('0');
            hexString.append(hex);
        }
        restituire hexString.toString();
    }
    catch(Eccezione e) {
        lanciare la nuova RuntimeException(e);
    }
}

pubblico vuoto statico principale(String args[]){
    provare{
        Connessione     con=DriverManager.getConnection("jdbc:sqlite:C://memo/sqlite-tools-
win32-x86-3310100/trans.sqlite3");
        //Connessione al server Redis su localhost
        Jedis = nuovo Jedis ("localhost");
        jedis.auth ("memo1234");
        Dichiarazione stmt=con.createStatement();
        Stringa sql = "SELEZIONA * Dal birrificio";
    }
}

```

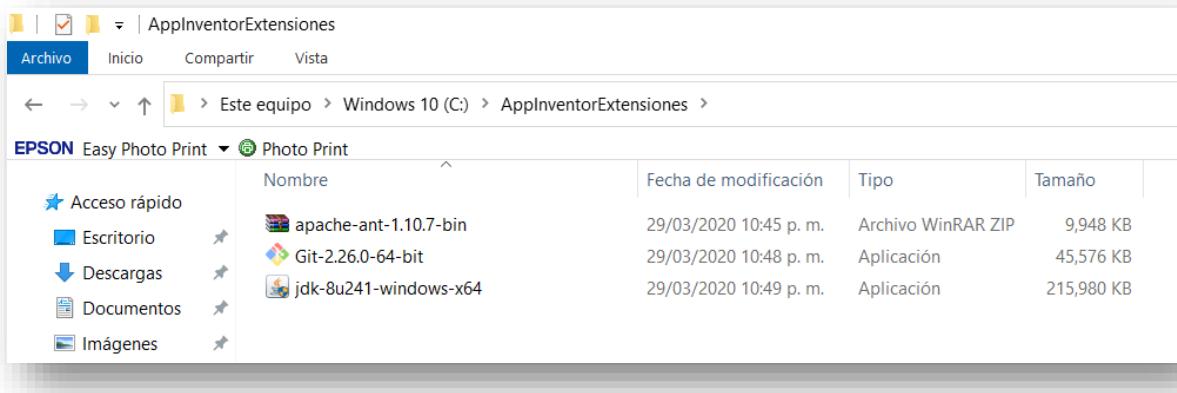
```
RisultatoSet rs=stmt.executeQuery(sql);
mentre(rs.next())) {
    transactions.add(rs.getString(2));
    jedis.set("LATAM:"+String.valueOf(rs.getInt(1)),"[\""+\"
\\""+"\""+rs.getString(2)+"\
\\""+"\""+\"
\\""+"\""+rs.getString(3)+"\""+","+"\""+\"
\\""+"\""+rs.getString(4)+"\""+"));
}
jedis.set("LATAM:merkleroot", getMerkleRoot());
System.out.println("Numero di elementi ArrayList: "+treeLayer.size());
con .close();
}catch(Eccezione e){ System.out.println(e);}

}
}
```

26. Allegato "Mini BlocklyChain per sviluppatori".

In questo allegato esamineremo la configurazione, l'installazione e l'uso di base di come generare moduli esterni basati sul linguaggio di programmazione Java per l'ambiente Blockly e saremo in grado di creare moduli specializzati per ogni business case e inserire funzionalità nel Mini BlocklyChain System o aggiungere altre funzionalità alla nostra applicazione mobile.

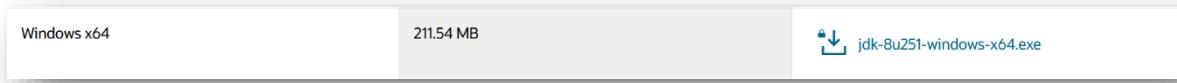
Abbiamo creato una directory nel nostro sistema Windows chiamata "AppInventorExtensions" e questa scaricherà i seguenti pacchetti software pubblici.



1.- Stiamo per scaricare l'ultima versione del **JDK (Java Development Kit)**

esempio: jdk-8u251-windows-x64.exe (211 MB)

<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>



2.- **Apache Ant** library che usa JAVA per costruire applicazioni, <http://ant.apache.org/bindownload.cgi>, nel mio caso ho scaricato Ant 1.10.8 (Binary Distributions) (apache-ant-1.10.8-bin.zip). Ci possono essere versioni più avanzate.

1.9.15 release - requires minimum of Java 5 at runtime

- 1.9.15 .zip archive: [apache-ant-1.9.15-bin.zip \[PGP\] \[SHA512\]](#)
- 1.9.15 .tar.gz archive: [apache-ant-1.9.15-bin.tar.gz \[PGP\] \[SHA512\]](#)
- 1.9.15 .tar.bz2 archive: [apache-ant-1.9.15-bin.tar.bz2 \[PGP\] \[SHA512\]](#)

1.10.8 release - requires minimum of Java 8 at runtime

- 1.10.8 .zip archive: [apache-ant-1.10.8-bin.zip \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.gz archive: [apache-ant-1.10.8-bin.tar.gz \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.bz2 archive: [apache-ant-1.10.8-bin.tar.bz2 \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.xz archive: [apache-ant-1.10.8-bin.tar.xz \[PGP\] \[SHA512\]](#)

Old Ant Releases

Older releases of Ant can be found [here](#). We highly recommend to not use those releases but upgrade to Ant's [latest](#) release.

3.- Abbiamo installato il Git Bash dal vostro sito <https://git-scm.com/download/win>

Downloading Git



You are downloading the latest (**2.27.0**) **64-bit** version of **Git for Windows**. This is the most recent [maintained build](#). It was released **9 days ago**, on **2020-06-01**.

[Click here to download manually](#)

Other Git for Windows downloads

[Git for Windows Setup](#)

[32-bit Git for Windows Setup](#).

[64-bit Git for Windows Setup.](#)

[Git for Windows Portable \("thumbdrive edition"\)](#)

[32-bit Git for Windows Portable.](#)

[64-bit Git for Windows Portable.](#)

The current source code release is version **2.27.0**. If you want the newer version, you can build it from [the source code](#).

4.- Decomprimere "Apache Ant." Quando avete finito di decomprimere, potete farlo ad esempio in una doppia cartella:

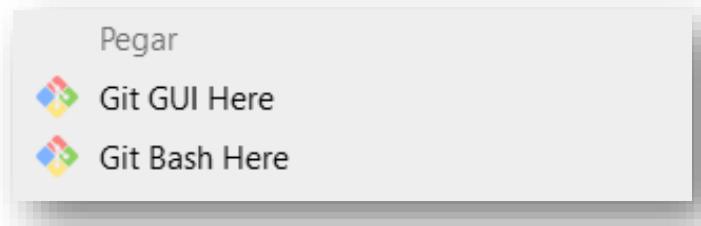
C:\AppInventorExtensions\apache-ant-1.10.8-bin\apache-ant-1.10.8-bin

- Cambia in C:\Apache-ant-1.10.8-bin

Nombre	Fecha de modificación	Tipo	Tamaño
bin	01/09/2019 11:43 a. m.	Carpeta de archivos	
etc	01/09/2019 11:43 a. m.	Carpeta de archivos	
lib	01/09/2019 11:43 a. m.	Carpeta de archivos	
manual	01/09/2019 11:43 a. m.	Carpeta de archivos	
CONTRIBUTORS	01/09/2019 11:43 a. m.	Archivo	7 KB
contributors	01/09/2019 11:43 a. m.	Documento XML	33 KB
fetch	01/09/2019 11:43 a. m.	Documento XML	14 KB
get-m2	01/09/2019 11:43 a. m.	Documento XML	5 KB
INSTALL	01/09/2019 11:43 a. m.	Archivo	1 KB
KEYS	01/09/2019 11:43 a. m.	Archivo	94 KB
LICENSE	01/09/2019 11:43 a. m.	Archivo	15 KB
NOTICE	01/09/2019 11:43 a. m.	Archivo	1 KB
patch	01/09/2019 11:43 a. m.	Documento XML	2 KB
README	01/09/2019 11:43 a. m.	Archivo	5 KB
WHATSNEW	01/09/2019 11:43 a. m.	Archivo	250 KB

5.- Abbiamo installato Git Bash. Abbiamo lasciato tutto di default nell'installazione.

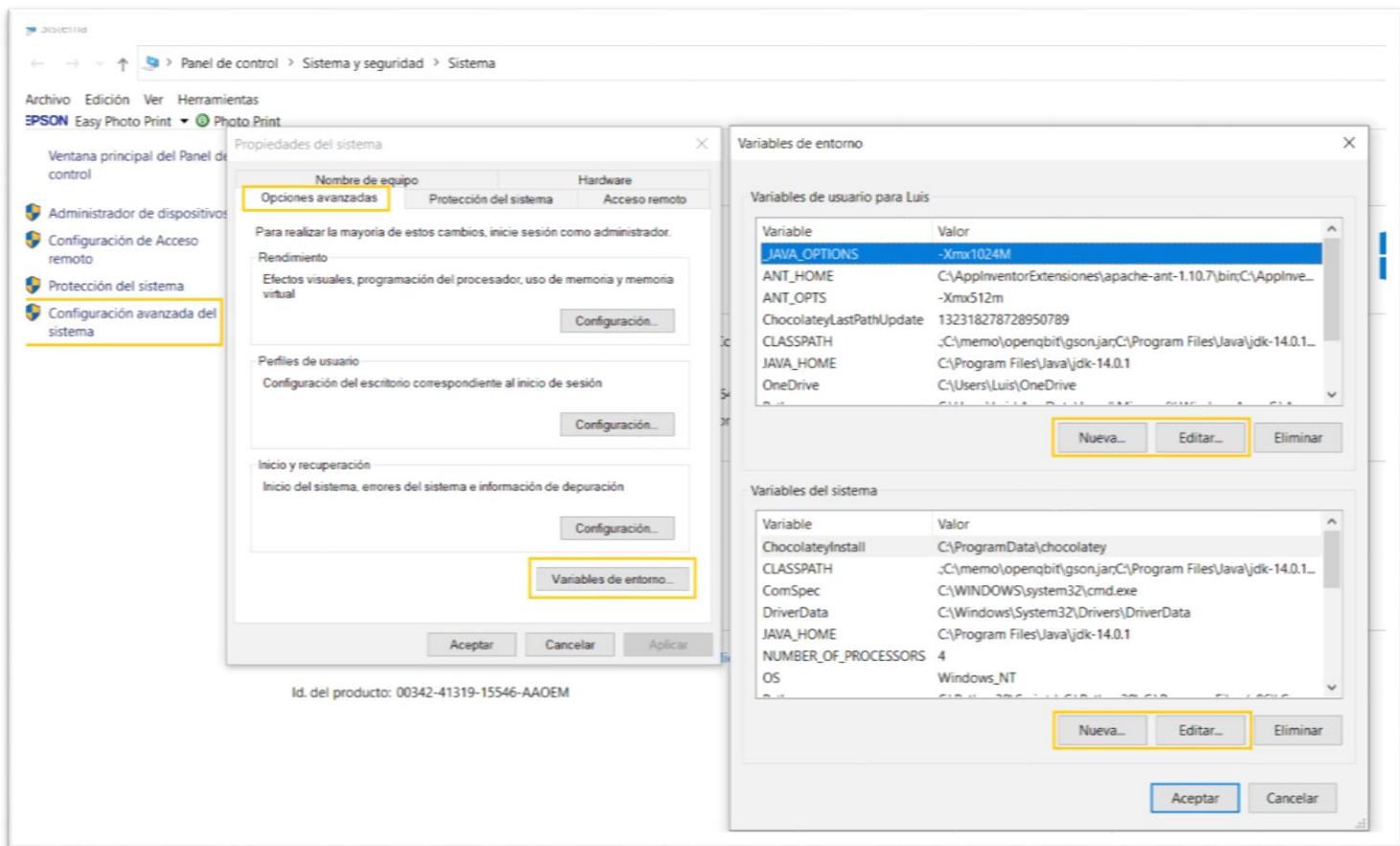
- In seguito possiamo vedere che ha un link nel pulsante di avvio di Windows, cliccando sul pulsante sinistro del browser dei file.



6.- Abbiamo installato il Java SE Development Kit. A seconda della versione di Windows sarà probabilmente necessario riavviare il computer.

Variabili d'ambiente del sistema Windows. Creeremo le variabili ambientali. Per fare questo lo faremo:

Pannello di controllo -> Sistema -> Impostazioni di sistema avanzate -> Opzioni avanzate -> Variabili di ambiente.



A seconda che si voglia inserire una nuova variabile o modificare una esistente, si preme il pulsante "Nuovo..." o "Modifica...".

Per aggiungere indirizzi a quelli già stabiliti, li separiamo per punto e virgola;

Nella sezione Variabili utente per John, abbiamo impostato questi nuovi...

JAVA_OPTIONS vi mettiamo di Valore -Xmx1024m

ANT_HOME abbiamo messo di Valore C:\AppInventorExtensions\apache-ant-1.10.8-bin [cioè la cartella dove abbiamo decompresso apache-ant]

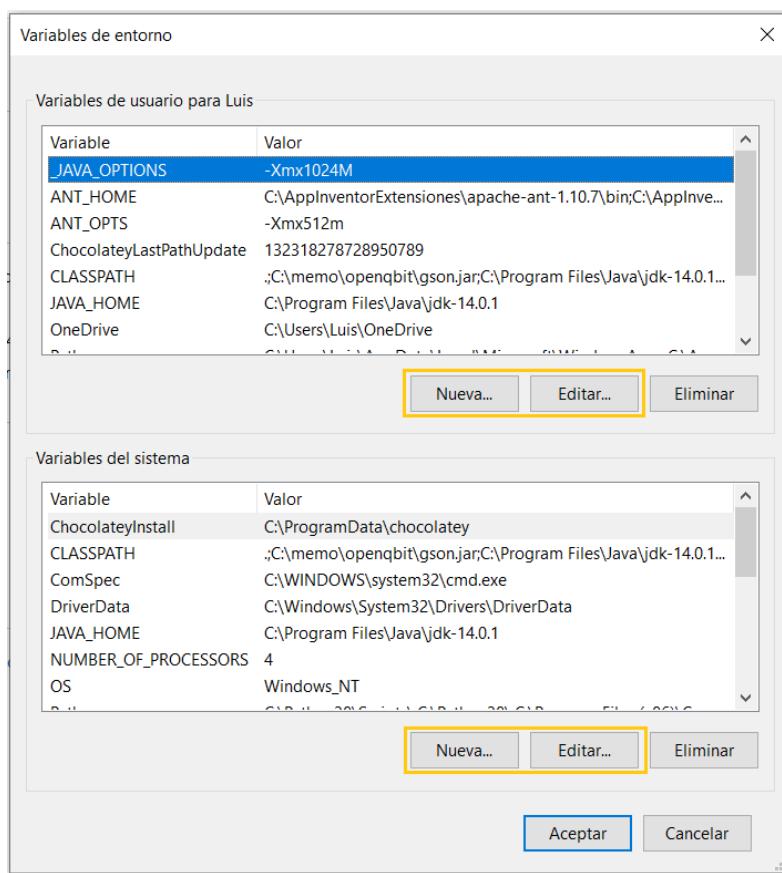
ANT_OPTS abbiamo messo di Valore -Xmx256M

JAVA_HOME è impostato su C:\Program Files\jdk1.8.0_131 [Se ha un altro valore, modificarlo. Si noti che è jdk NON jdr].

CLASSPATH mettiamo di Valore %ANT_HOME%\lib;%JAVA_HOME%\lib

In PATH abbiamo aggiunto ;%ANT_HOME%\bin;%JAVA_HOME%\bin [Si noti che ;inizia con un punto e virgola; da aggiungere a quelli già presenti].

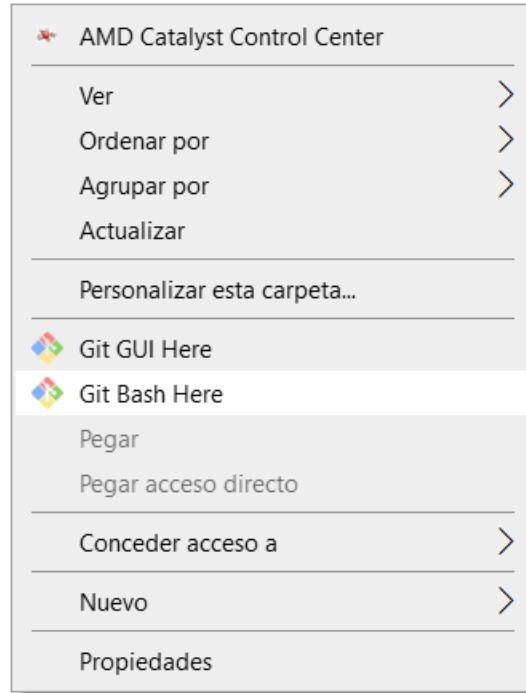
NOTA: Le variabili sono separate l'una dall'altra da un punto e virgola: variabile-uno; variabile-n; variabile-n; variabile-n+1



7.- Creazione del clone di App Inventor nel nostro computer

Creeremo una copia "clone" di App Inventor sul nostro server (PC), la scaricheremo direttamente da Internet e creeremo quella copia.

Per fare questo, useremo l'applicazione **Git Bash** e cliccheremo su di essa per aprire un terminale.



Eseguiamo il comando sul terminale di Git Bash:

```
$ git clone https://github.com/mit-cml/appinventor-sources.git
```

```
uis@DESKTOP-LLGPLR6 MINGW64 ~
git clone https://github.com/mit-cml/appinventor-sources.git
Cloning into 'appinventor-sources'...
remote: Counting objects: 41191, done.
remote: Total 41191 (delta 0), reused 0 (delta 0), pack-reused 41190
receiving objects: 100% (41191/41191), 553.30 MiB | 494.00 KiB/s, done.
resolving deltas: 100% (21999/21999), done.
Checking out files: 100% (1758/1758), done.
uis@DESKTOP-LLGPLR6 MINGW64 ~
```

Il sito dove si trova il deposito:

<https://github.com/mit-cml/appinventor-sources/>

Creerà una cartella chiamata appinventor-sources con il sorgente App Inventor.

Nombre	Fecha de modificación	Tipo	Tamaño
.github	30/03/2020 01:05 a. m.	Carpeta de archivos	
appinventor	03/05/2020 05:32 p. m.	Carpeta de archivos	
.gitmodules	30/03/2020 01:05 a. m.	Documento de tex...	1 KB
bootstrap	30/03/2020 01:05 a. m.	Shell Script	2 KB
LICENSE	30/03/2020 01:05 a. m.	Archivo	12 KB
README.md	30/03/2020 01:05 a. m.	Archivo MD	11 KB
sample-	30/03/2020 01:05 a. m.	Documento de tex...	1 KB
Vagrantfile	30/03/2020 01:05 a. m.	Archivo	1 KB

Abbiamo creato la seguente directory nel percorso C: Utenti - Appinventor-sourcesv-babkup - Appinventor-components -rc-openqbit

All'interno abbiamo copiato il nostro seguente programma di test chiamato OpenQbitQRNGch.java

Nombre	Fecha de modificación	Tipo	Tamaño
OpenQbitQRNGch	11/06/2020 01:39 a. m.	Archivo JAVA	5 KB

Creazione dell'estensione.

RISPETTA I TAPPI E I MINUSCOLI, non è lo stesso Ciao e Ciao.

Non mettere accenti. Siamo pronti a fare la nostra prima estensione, sarà il generatore di numeri casuali quantistici.

Usiamo un Text Editor, Notepad++, creiamo un file chiamato OpenQbitQRNGch.java che genera numeri quantici casuali con il seguente codice:

```
// Questa estensione è utilizzata per l'uscita del generatore di numeri casuali quantistici QRNG Switzerland API.

pacchetto com.openqbit. OpenQbitQRNGch;
importare
com.google.appinventor.components.annotations.DesignerComponent;
importare com.google.appinventor.components.annotations.DesignerProperty;
importare com.google.appinventor.components.annotations.PropertyCategory;
importare com.google.appinventor.components.annotations.SimpleEvent;
importare com.google.appinventor.components.annotations.SimpleFunction;
importare com.google.appinventor.components.annotations.SimpleObject;
importare com.google.appinventor.components.annotations.SimpleProperty;
importare com.google.appinventor.components.common.ComponentCategory;
importare com.google.appinventor.components.common.PropertyTypeConstants;
importare com.google.appinventor.components.runtime.util.mediaUtil;
importare com.google.appinventor.components.runtime.*;
importare java.io.*;

@DesignerComponent(versione = OpenQbitQRNGch.VERSION,
    descrizione = "API Quantum Random Number Generator. Numeri casuali quantistici come servizio, QRNGaaS, web API per il generatore di numeri casuali quantistici di Quantis sviluppato dall'azienda svizzera - " +
    "Guillermo Vidal - OpenQbit.com",
    categoria = ComponentCategory.EXTENSION
    non visibile = vero,
    iconName = "http://www.pinntar.com/logoQbit.png")
@SempliceOggetto(esterno = vero)

classe pubblica OpenQbitQRNGch estende Android NonvisibleComponent
implementa Component {

    VERSIONE finale statica pubblica finale int VERSIONE = 1;
    pubblico statico finale Stringa finale DEFAULT_TEXT_1 = "";
    contenitore privato ComponentContainer;
    privato Stringa testo_1 = "";

    pubblico OpenQbitQRNGch(ComponentContainer container) {
        super(container.$form());
        this.container = contenitore;
    }

    // Stabilimento delle proprietà.
    //@DesignerProperty(editorType =
    PropertyTypeConstants.PROPERTY_TYPE_STRING, defaultValue =
    OpenQbitQRNGch.DEFAULT_TEXTO_1 + "")
    //@SimpleProperty(descrizione = "API Get Quantum Random Number Generator,
    numero casuale tra 0 e 1. Inserire la variabile *quantità* dei numeri da generare")

    @SimpleFunction(description = "API Get Quantum Random Number Generator,
    numero casuale tra 0 e 1. Inserire il numero intero variabile *quantità*
    dei numeri da generare")
}
```

```
pubblico Stringa APIGetQRNGdecimale(Stringa qty) tiri Eccezione {
    String url = "http://random.openqu.org/api/rand?size=" + qty;
    Comando String[] = { "curl", url };

    ProcessBuilder processo = nuovo ProcessBuilder(comando);
    Processo p;
    p = processo.start();
    BufferedReader reader = nuovo BufferedReader(nuovo
InputStreamReader(p.getInputStream()));
    StringBuilder builder = nuovo StringBuilder();
    Linea di stringa = nulla;
    mentre ((linea = reader.readLine()) != null) {
        costruttore.append(linea);

        builder.append(System.getProperty("line.separator"));
    }
    Risultato stringa = builder.toString();
    //System.out.print(risultato);
    risultato di ritorno;

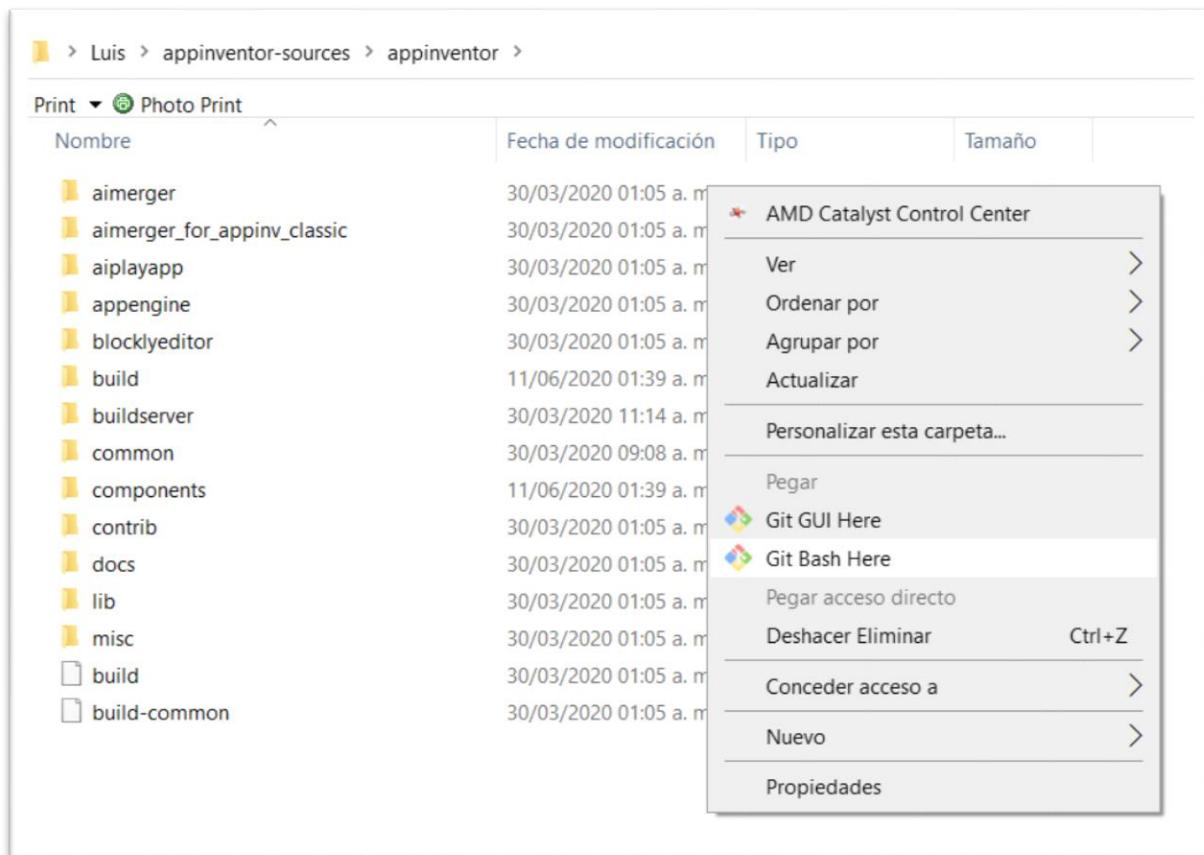
}
@SimpleFunction(description = "API Get Quantum Random Number Generator,
numero casuale tra un intervallo di numeri da min a max. Inserire il
numero intero variabile *quantità* di numeri per generare un intervallo
tra un numero minimo *min* e un numero massimo *max*)
pubblico String APIGetQRNGinteger(String qty, String min, String max)
tiri Eccezione {
    String url = "http://random.openqu.org/api/randint?size=" + qty +
"&min=" + min + "&max=" + max
    Comando String[] = { "curl", url };

    ProcessBuilder processo = nuovo ProcessBuilder(comando);
    Processo p;
    p = processo.start();
    BufferedReader reader = nuovo BufferedReader(nuovo
InputStreamReader(p.getInputStream()));
    StringBuilder builder = nuovo StringBuilder();
    Linea di stringa = nulla;
    mentre ((linea = reader.readLine()) != null) {
        costruttore.append(linea);

        builder.append(System.getProperty("line.separator"));
    }
    Risultato stringa = builder.toString();
    //System.out.print(risultato);
    risultato di ritorno;

}
}
```

Eseguiamo il **Git Bash** posizionandoci sul seguente percorso: **C:\Luis\appinventor-sources\appinventor**. In questa directory, clicchiamo con il tasto sinistro del mouse e selezioniamo il terminale **Git Bash**.



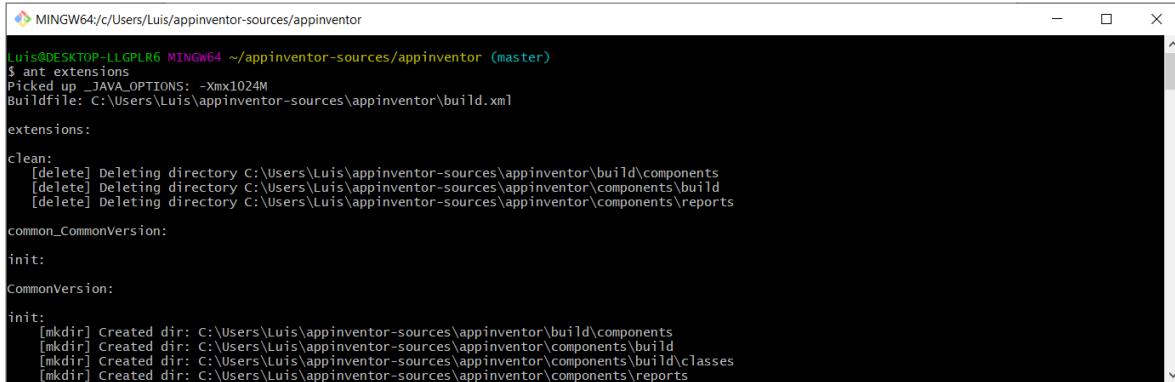
Il terminale Git bash sarà posizionato nella seguente directory:

C:\\\\x22C:\\x22Users\\x22Luis\\x22appinventor-sources\\x22appinventor\\x22(master)\\x22

```
MINGW64:/c/Users/Luis/appinventor-sources/appinventor
Luis@DESKTOP-LLGPLR6 MINGW64 ~ /appinventor-sources/appinventor (master)
$ |
```

Nel terminale Git Bash ed eseguire il seguente comando:

\$ estensione formica



```
Luis@DESKTOP-LLGPLR6 MINGW64 ~/appinventor-sources/appinventor (master)
$ ant extensions
Picked up _JAVA_OPTIONS: -Xmx1024M
Buildfile: C:\Users\Luis\appinventor-sources\appinventor\build.xml

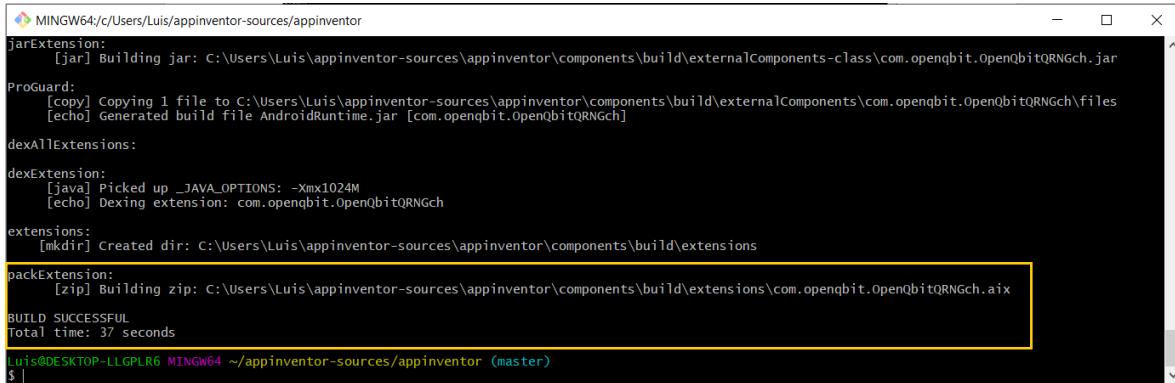
extensions:
clean:
[delete] Deleting directory C:\Users\Luis\appinventor-sources\appinventor\build\components
[delete] Deleting directory C:\Users\Luis\appinventor-sources\appinventor\components\build
[delete] Deleting directory C:\Users\Luis\appinventor-sources\appinventor\components\reports

common_CommonVersion:
init:
CommonVersion:
init:
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\build\components
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\build
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\build\classes
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\reports
```

Se tutto è andato bene, ce la faremo: COSTRUIRE CON SUCCESSO.

La nostra estensione è stata creata...

NOTA: il contenuto della cartella delle estensioni viene cancellato e ricreato ogni volta che realizziamo un'estensione per formiche.



```
Luis@DESKTOP-LLGPLR6 MINGW64 ~/appinventor-sources/appinventor (master)
jarExtension:
[jar] Building jar: C:\Users\Luis\appinventor-sources\appinventor\components\build\externalComponents-class\com.openqbit.OpenQbitQRNGch.jar

ProGuard:
[copy] Copying 1 file to C:\Users\Luis\appinventor-sources\appinventor\components\build\externalComponents\com.openqbit.OpenQbitQRNGch\files
[echo] Generated build file AndroidRuntime.jar [com.openqbit.OpenQbitQRNGch]

dexAllExtensions:

dexExtension:
[java] Picked up _JAVA_OPTIONS: -Xmx1024M
[echo] Dexing extension: com.openqbit.OpenQbitQRNGch

extensions:
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions

packExtension:
[zip] Building zip: C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions\com.openqbit.OpenQbitQRNGch.aix

BUILD SUCCESSFUL
Total time: 37 seconds

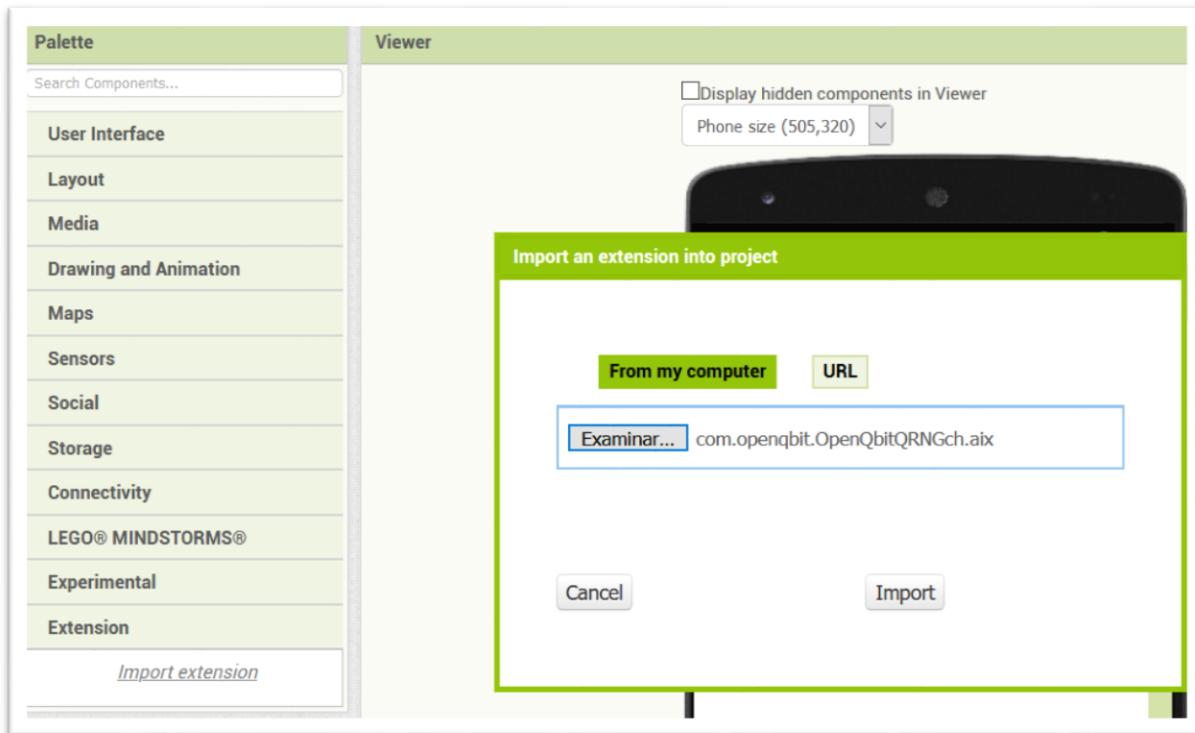
Luis@DESKTOP-LLGPLR6 MINGW64 ~/appinventor-sources/appinventor (master)
$ |
```

Andiamo a quella cartella:

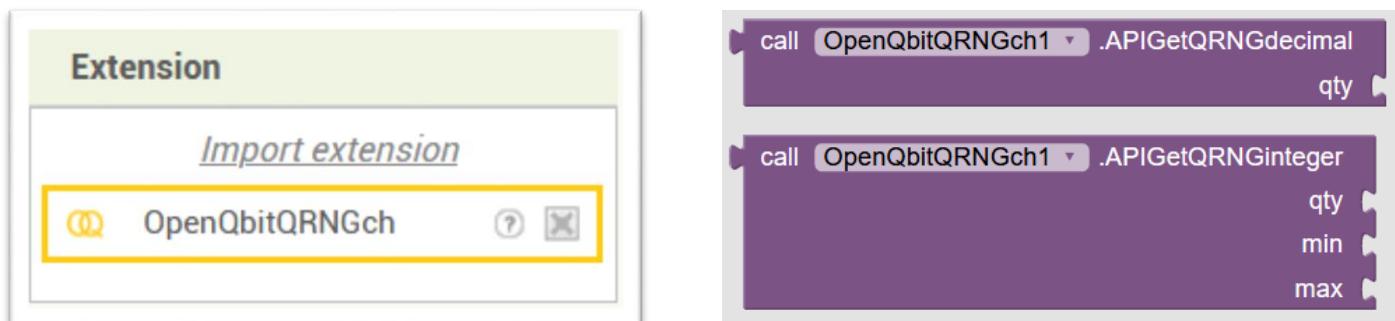
e copiare il file com.openqbit.OpenQbit.OpenQbitQRNGch.aix, questo file è la nostra estensione.

Avendo già un account in App Inventor o in un altro sistema Blockly siamo entrati per aggiungere la nuova estensione e testarla.

Andate in basso dove si trova l'opzione Estensione e cliccate su "Importa estensione":



Premiamo il pulsante "Importa". Ora abbiamo i blocchi (**APIGetQRNGdecimal**) e (**APIGetQRNGinteger**) da utilizzare:



Abbiamo già la nostra prima estensione creata e funzionale.

27. Allegato "Contratti intelligenti BlocklyCode".

I BlocklyCodes sono programmi realizzati in linguaggio java. L'estensione per creare questo tipo di programma comunemente chiamato "Smart Contracts" è un modo per elaborare gli accordi tra gli utenti (aziende o persone).

Questo blocco (**BlocklyCode**), viene implementato in un programma che ha già dei parametri stabiliti e che a seconda del tipo di accordi che potrebbero essere eseguiti dal sistema Mini BlocklyChain in modo automatico quando i locali che regolano il "Contratto Smart" sono soddisfatti.

I parametri da considerare come parametri suggeriti o parametri di ingresso 'ingressi' sono

Data di scadenza

Data di riferimento

Tempo di scadenza

Tempo di riferimento

Attività di riferimento

Totale immobilizzazioni

Attività variabili

Membri a contratto

Dati confermati (Stringa)

Dati confermati (Nome del file)

Evento variabile

Evento fisso

Convalida del documento o dei documenti

Convalida delle stringhe

Firma valida

Intervallo definito (intero)

Intervallo decimale

Minimo stabilito

Stabilire il massimo

Prima di iniziare ad utilizzare il blocco (**BlocklyCode**) dobbiamo prima installare il sistema che eseguirà l'esecuzione degli "Smart Contracts", questo viene fatto installando nel terminale di Termux OpenJDK e OpenJRE.

OpenJDK (Open Java Development Kit di sviluppo Java). - È lo strumento per sviluppare programmi in java, contiene le librerie, il compilatore e la JVM (Java Virtual Machine).

OpenJRE (Open Java Runtime Environment). - Questo è lo strumento per eseguire solo programmi java. Contiene la JVM (Java Virtual Machine).

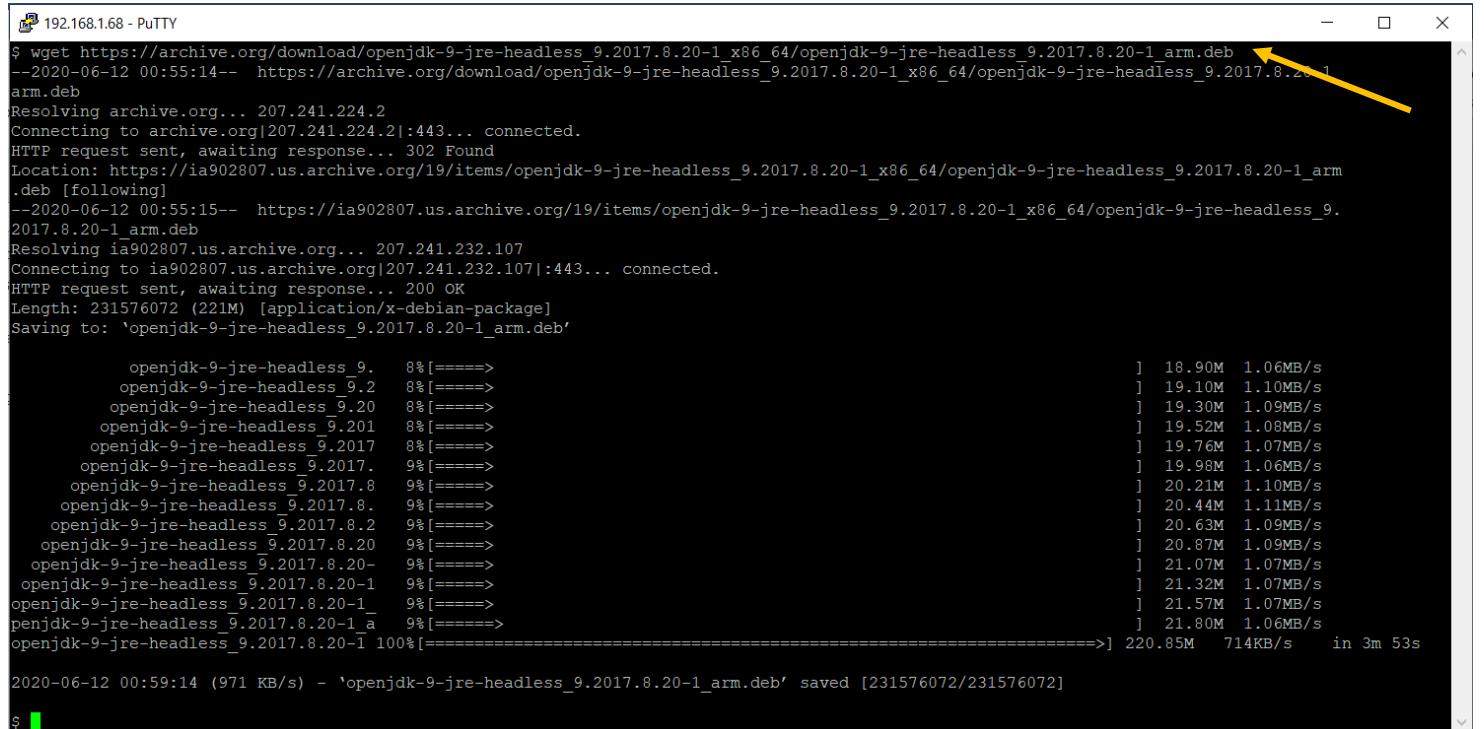
Procediamo con l'installazione di OpenJRE e OpenJDK nel terminale Termux dei nodi che formano il sistema Mini BlocklyChain. Abbiamo installato le camere

\$ apt install - e wget



Abbiamo scaricato l'OpenJRE

\$ wget https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb



```
$ wget https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
--2020-06-12 00:55:14-- https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
Resolving archive.org... 207.241.224.2
Connecting to archive.org|207.241.224.2|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://ia902807.us.archive.org/19/items/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb [following]
--2020-06-12 00:55:15-- https://ia902807.us.archive.org/19/items/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
Resolving ia902807.us.archive.org... 207.241.232.107
Connecting to ia902807.us.archive.org|207.241.232.107|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 231576072 (221M) [application/x-debian-package]
Saving to: 'openjdk-9-jre-headless_9.2017.8.20-1_arm.deb'

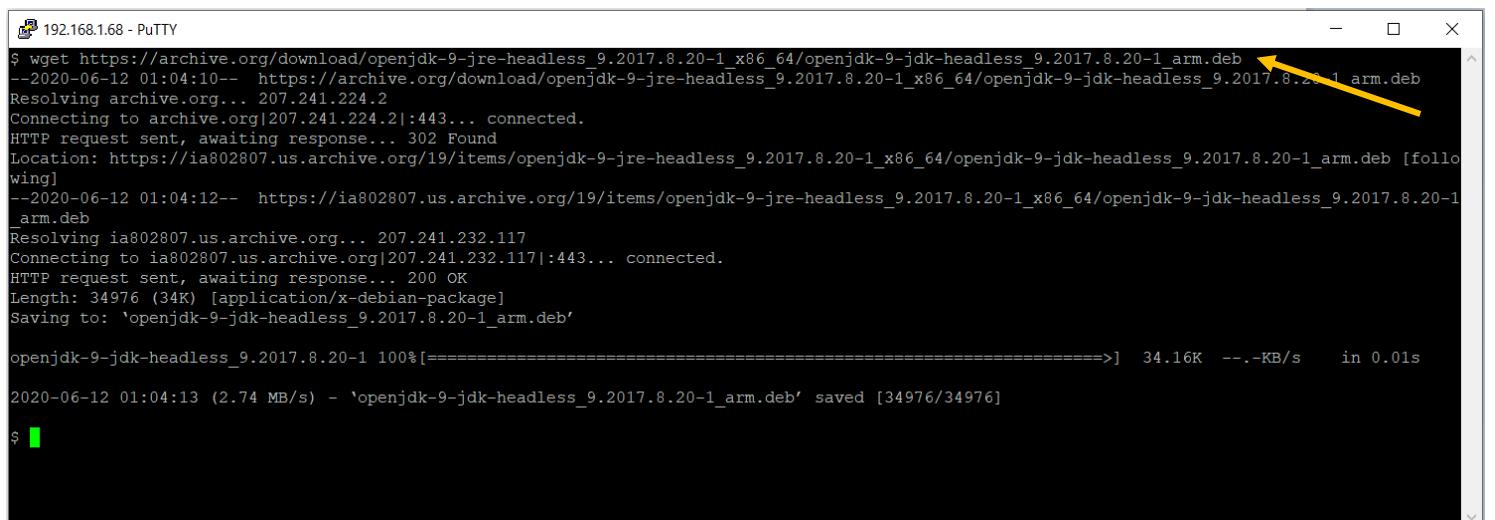
openjdk-9-jre-headless_9. 8%[=====] ] 18.90M 1.06MB/s
openjdk-9-jre-headless_9.2 8%[=====] ] 19.10M 1.10MB/s
openjdk-9-jre-headless_9.20 8%[=====] ] 19.30M 1.09MB/s
openjdk-9-jre-headless_9.201 8%[=====] ] 19.52M 1.08MB/s
openjdk-9-jre-headless_9.2017 8%[=====] ] 19.76M 1.07MB/s
openjdk-9-jre-headless_9.2017. 9%[=====] ] 19.98M 1.06MB/s
openjdk-9-jre-headless_9.2017.8 9%[=====] ] 20.21M 1.10MB/s
openjdk-9-jre-headless_9.2017.8. 9%[=====] ] 20.44M 1.11MB/s
openjdk-9-jre-headless_9.2017.8.2 9%[=====] ] 20.63M 1.09MB/s
openjdk-9-jre-headless_9.2017.8.20 9%[=====] ] 20.87M 1.09MB/s
openjdk-9-jre-headless_9.2017.8.20- 9%[=====] ] 21.07M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1 9%[=====] ] 21.32M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1_ 9%[=====] ] 21.57M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1_a 9%[=====] ] 21.80M 1.06MB/s
openjdk-9-jre-headless_9.2017.8.20-1 100%[=====] 220.85M 714KB/s in 3m 53s

2020-06-12 00:59:14 (971 KB/s) - 'openjdk-9-jre-headless_9.2017.8.20-1_arm.deb' saved [231576072/231576072]

$
```

Abbiamo scaricato l'OpenJDK

\$ wget https://archive.org/download/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb



```
$ wget https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
--2020-06-12 01:04:10-- https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
Resolving archive.org... 207.241.224.2
Connecting to archive.org|207.241.224.2|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://ia802807.us.archive.org/19/items/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb [following]
--2020-06-12 01:04:12-- https://ia802807.us.archive.org/19/items/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
Resolving ia802807.us.archive.org... 207.241.232.117
Connecting to ia802807.us.archive.org|207.241.232.117|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 34976 (34K) [application/x-debian-package]
Saving to: 'openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb'

openjdk-9-jdk-headless_9.2017.8.20-1 100%[=====] 34.16K ---KB/s in 0.01s

2020-06-12 01:04:13 (2.74 MB/s) - 'openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb' saved [34976/34976]

$
```

Eseguiamo l'installazione dal terminale Termux di OpenJDK e OpenJRE:

```
$ apt install -e ./openjdk-9-jre-headless_9.2017.8.20-1_arm.deb ./openjdk-9-jdk-
headless_9.2017.8.20-1_arm.deb
```



```
$ ls
openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
$ apt install -y ./openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb ./openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'openjdk-9-jdk-headless' instead
of './openjdk-9-jdk-headless_9.2017.8.20-1_arm.
deb'
Note, selecting 'openjdk-9-jre-headless' instead
of './openjdk-9-jre-headless_9.2017.8.20-1_arm.
deb'
The following additional packages will be instal
led:
  ca-certificates-java freetype libpng
The following NEW packages will be installed:
  ca-certificates-java freetype libpng
  openjdk-9-jdk-headless
  openjdk-9-jre-headless
0 upgraded, 5 newly installed, 0 to remove and 0
not upgraded.
Need to get 668 kB/232 MB of archives.
After this operation, 376 MB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-packa
ges-24 stable/main arm ca-certificates-java all
20200101 [110 kB]
Get:2 https://dl.bintray.com/termux/termux-packa
ges-24 stable/main arm libpng arm 1.6.37-2 [190
kB]
Get:3 https://dl.bintray.com/termux/termux-packa
ges-24 stable/main arm freetype arm 2.10.2 [368
kB]
Get:4 /data/data/com.termux/files/home/openjdk/o
penjdk-9-jre-headless_9.2017.8.20-1_arm.deb open
jdk-9-jre-headless arm 9.2017.8.20-1 [232 MB]
Get:5 /data/data/com.termux/files/home/openjdk/o
penjdk-9-jdk-headless_9.2017.8.20-1_arm.deb open
jdk-9-jdk-headless arm 9.2017.8.20-1 [35.0 kB]
```



```
$ ls
openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
Get:5 /data/data/com.termux/files/home/openjdk/o
penjdk-9-jdk-headless_9.2017.8.20-1_arm.deb open
jdk-9-jdk-headless arm 9.2017.8.20-1 [35.0 kB]
Fetched 668 kB in 23s (28.0 kB/s)
Selecting previously unselected package ca-certifi
cates-java.
(Reading database ... 16939 files and directo
ries currently installed.)
Preparing to unpack .../ca-certificates-java_202
00101_all.deb ...
Unpacking ca-certificates-java (20200101) ...
Selecting previously unselected package libpng.
Preparing to unpack .../libpng_1.6.37-2_arm.deb
...
Unpacking libpng (1.6.37-2) ...
Selecting previously unselected package freetype
.
Preparing to unpack .../freetype_2.10.2_arm.deb
...
Unpacking freetype (2.10.2) ...
Selecting previously unselected package openjdk-
9-jre-headless.
Preparing to unpack .../openjdk-9-jre-headless_9
.2017.8.20-1_arm.deb ...
Unpacking openjdk-9-jre-headless (9.2017.8.20-1)
...
Selecting previously unselected package openjdk-
9-jdk-headless.
Preparing to unpack .../openjdk-9-jdk-headless_9
.2017.8.20-1_arm.deb ...
Unpacking openjdk-9-jdk-headless (9.2017.8.20-1)
...
Setting up libpng (1.6.37-2) ...
Setting up freetype (2.10.2) ...
Setting up ca-certificates-java (20200101) ...
Setting up openjdk-9-jre-headless (9.2017.8.20-1)
...
Setting up openjdk-9-jdk-headless (9.2017.8.20-1)
...
$
```

Da quando abbiamo finito l'installazione dell'ambiente OpenJDK e OpenJRE. Queste installazioni contengono la JVM (Java Virtual Machine) che sarà l'ambiente che eseguirà il BlocklyCode.

Ora installeremo un editor per creare file online, useremo l'editor chiamato **vi** eseguiremo il seguente comando:

\$ apt install vim

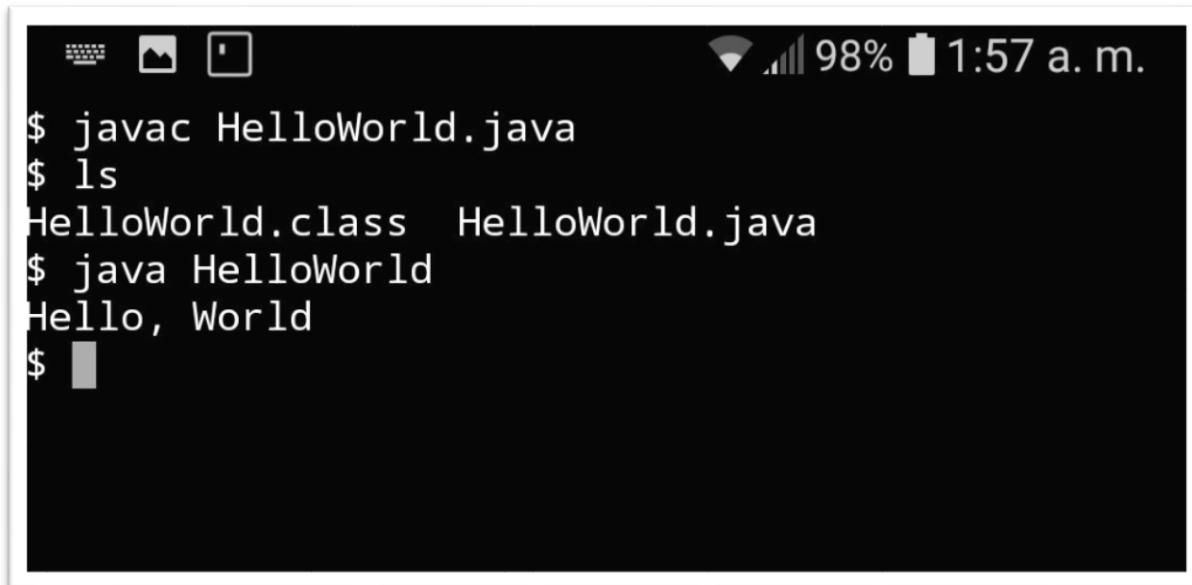
Ora proviamo a compilare un file di prova e a eseguirlo. Creiamo nel terminale Termux con l'editor **vi** e scriviamo il codice java di un "Hello World", e lo salviamo nel file HelloWorld.java

```
classe pubblica HelloWorld {  
  
    pubblico vuoto statico principale(String[] args) {  
        // Stampa "Hello, World" sulla finestra del terminale.  
        System.out.println ("Ciao, Mondo");  
    }  
  
}
```

Poi si esegue il seguente comando nel terminale Termux per la compilazione e poi l'esecuzione del programma:

\$ javac HelloWorld.java (Dopo l'esecuzione, viene creato un file bytecode HelloWorld.class)

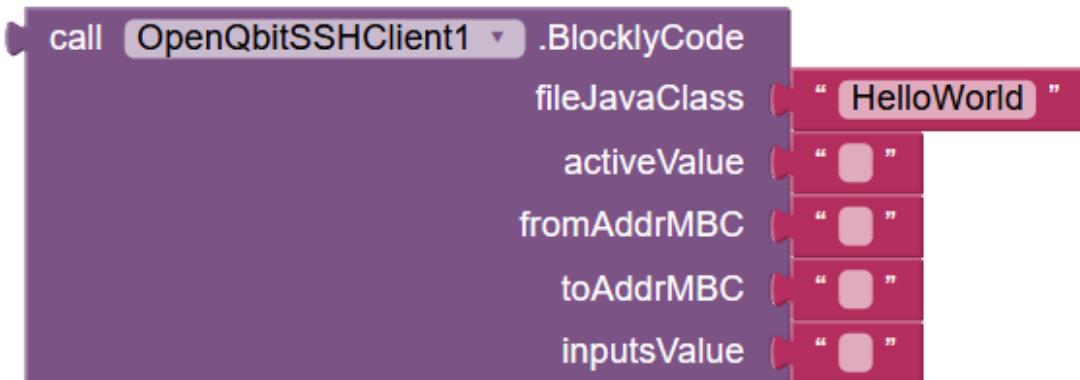
\$ java HelloWorld (Dopo aver eseguito e stampato l'annuncio, Hello World)



The screenshot shows a Termux terminal window on a mobile device. The status bar at the top right indicates a battery level of 98% and the time as 1:57 a.m. The terminal window displays the following command-line session:

```
$ javac HelloWorld.java  
$ ls  
HelloWorld.class  HelloWorld.java  
$ java HelloWorld  
Hello, World  
$
```

Utilizzando il blocco **(BlocklyCode)** questo blocco si trova nell'estensione



(ConnectorSSHClient).

Nel blocco precedente si può vedere come verrà eseguito il file HelloWorld.class, che è già stato compilato e posizionato nella directory base dell'utente.

Si tratta di un blocco in cui è possibile eseguire un programma già compilato in java, che è comunemente conosciuto nell'ambiente di sviluppo come file bytecode nella sua forma binaria.

Questo tipo di file è pronto per essere eseguito dalla Java Virtual Machine (JVM).

Ci sono due modi per creare un file bytecode o con estensione .CLASS, l'utente può crearlo nel suo PC comodamente o il suo simile può essere fatto anche nel cellulare, ricordiamo che abbiamo già installato l'ambiente per compilare JDK (Java Development Kit) prima, anche se sarà meno efficiente perché la limitazione della tastiera conterà, anche in caso di scelta dell'ambiente Android dovrà tenere conto che le librerie sono limitate ad OpenJDK che è stato installato.

I parametri di ingresso sono aperti in <inputsValue> e gli altri fissi sono quelli di activeValue, fromaddrMBC e toAddrMBC.

Nel caso di test di esecuzione possono essere lasciati vuoti senza contenuto e solo il file del bytecode verrà eseguito senza parametri.

Il blocco precedente è come la riga di comando successiva che viene eseguita:

\$ java HelloWorld

I programmi sviluppati per BlocklyCode saranno soggetti ad ogni particolare ambiente di progettazione e potranno essere controllati ed eseguiti di routine con l'agente "cron" e condivisi con syncthingmanager.

28. Allegato "OpenQbit Quantum Computing".

Come funziona il calcolo quantistico? ⁽²⁾

La trasformazione digitale sta portando il mondo a un cambiamento più rapido che mai: ci credereste che l'era digitale sta per finire? **L'alfabetizzazione digitale** è già stata identificata come un'area in cui la conoscenza aperta e le opportunità accessibili di apprendere la tecnologia sono urgenti per affrontare le lacune nello sviluppo sociale ed economico. Imparare dai concetti chiave dell'era digitale diventerà ancora più critico con l'imminente arrivo di un'altra nuova ondata tecnologica in grado di trasformare i modelli esistenti con velocità e potenza sorprendenti: **le tecnologie quantistiche**.

In questo articolo confrontiamo i concetti di base del calcolo tradizionale e del calcolo quantistico; e iniziamo anche ad esplorare la loro applicazione in altre aree correlate.

Cosa sono le tecnologie quantistiche?

Nel corso della storia, gli esseri umani hanno sviluppato la tecnologia così come hanno capito come funziona la natura attraverso la scienza. Tra il 1900 e il 1930, lo studio di alcuni fenomeni fisici non ancora ben compresi ha dato origine ad una nuova teoria fisica, la **Meccanica Quantistica**. Questa teoria descrive e spiega il funzionamento del mondo microscopico, l'habitat naturale di molecole, atomi o elettroni. Grazie a questa teoria, non solo è stato possibile spiegare questi fenomeni, ma è stato anche possibile comprendere che la realtà subatomica funziona in modo completamente contro-intuitivo, quasi magico, e che nel mondo microscopico avvengono eventi che non si verificano nel mondo macroscopico.

Queste **proprietà quantistiche** comprendono la sovrapposizione quantistica, l'entanglement quantistico e il teletrasporto quantistico.

- **La sovrapposizione quantistica** descrive come una particella può trovarsi in diversi stati contemporaneamente.
- **L'entanglement quantistico** descrive come due particelle così distanti tra loro possano essere correlate in modo tale che, interagendo con una, l'altra ne sia consapevole.
- **Il teletrasporto quantistico** usa l'entanglement quantistico per inviare informazioni da un luogo ad un altro nello spazio senza doverlo attraversare.

Le tecnologie quantistiche si basano su queste proprietà quantistiche di natura subatomica.

In questo caso, oggi la comprensione del mondo microscopico attraverso la Meccanica Quantistica ci permette di inventare e progettare tecnologie in grado di migliorare la vita delle persone. Ci sono molte e molto diverse tecnologie che utilizzano i fenomeni quantistici e alcune di esse, come i laser o la risonanza magnetica (MRI), sono con noi da più di mezzo secolo. Tuttavia, stiamo attualmente assistendo ad una rivoluzione tecnologica in settori

quali il calcolo quantistico, l'informazione quantistica, la simulazione quantistica, l'ottica quantistica, la metrologia quantistica, gli orologi quantistici o i sensori quantistici.

Cos'è il calcolo quantistico? Per prima cosa, bisogna capire l'informatica classica.



Caracter	Bits
7	111
A	01000001
\$	00100100
:)	0011101000101001

FIGURA 1.
Ejemplos de caracteres en lenguaje binario.

Per

capire come funzionano i computer quantistici, è conveniente spiegare prima di tutto come funzionano i computer che usiamo ogni giorno, che in questo documento chiameremo computer digitali o classici. Questi, come il resto dei dispositivi elettronici come i tablet o i telefoni cellulari, utilizzano i bit come unità fondamentali della memoria. Ciò significa che i programmi e le applicazioni sono codificati in bit, cioè in linguaggio binario di zeri e uno. Ogni volta che interagiamo con uno di questi dispositivi, ad esempio, premendo un tasto sulla tastiera, si creano, distruggono e/o modificano stringhe di zeri e uno all'interno del computer.

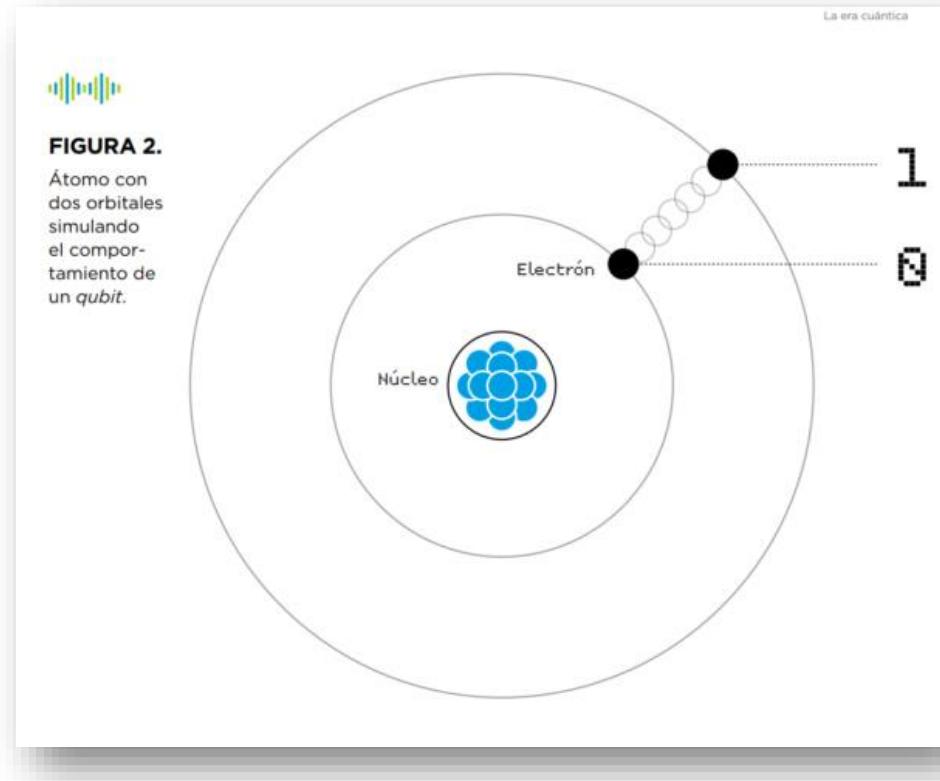
La domanda interessante è: quali sono questi zeri e quali sono fisicamente all'interno del computer? Gli stati zero e uno corrispondono alla corrente elettrica che circola, o meno, attraverso parti microscopiche chiamate transistor, che fungono da interruttori. Quando non scorre corrente, il transistor è "spento" e corrisponde al bit 0, e quando scorre è "acceso" e corrisponde al bit 1.

Più semplicemente, è come se i bit 0 e 1 corrispondessero a fori, in modo che un foro vuoto sia un bit 0 e un foro occupato da un elettrone sia un bit 1. Per questo motivo questi apparecchi sono chiamati elettronici. Come esempio, la figura 1 mostra la scrittura binaria di alcuni caratteri. Ora che abbiamo un'idea di come funzionano i computer di oggi, cerchiamo di capire come funzionano i quantum.

Da bit a qubits

L'unità fondamentale dell'informazione nel calcolo quantistico è il bit quantistico o qubit. I Qubit sono, per definizione, sistemi quantistici a due livelli - vedremo qui degli esempi - che, come i bit, possono essere a basso livello, che corrisponde ad uno stato di bassa eccitazione

o di energia definito come 0, o ad alto livello, che corrisponde ad uno stato di eccitazione superiore o definito come 1. Tuttavia, e qui sta la differenza fondamentale con il calcolo classico, i qubit possono anche trovarsi in uno qualsiasi degli infiniti stati intermedi tra 0 e 1, come ad esempio uno stato che è metà 0 e metà 1, o tre quarti di 0 e un quarto di 1.



Algoritmi quantistici, calcolo esponenzialmente più potente ed efficiente

Lo scopo dei computer quantistici è quello di sfruttare queste proprietà quantistiche dei *qubit*, come sistemi quantistici che sono, al fine di eseguire algoritmi quantistici che utilizzano la sovrapposizione e l'interleaving per fornire una potenza di elaborazione molto maggiore rispetto ai classici. È importante sottolineare che il vero cambiamento di paradigma non consiste nel fare la stessa cosa che fanno i computer digitali o classici -quelli attuali- ma più velocemente, come si può leggere in molti articoli, ma che gli algoritmi quantistici permettono di eseguire certe operazioni in un modo totalmente diverso che in molti casi si rivela più efficiente - cioè in molto meno tempo o utilizzando molto meno risorse computazionali.

Vediamo un esempio concreto di ciò che questo comporta. Immaginiamo di essere a Bogotá e vogliamo sapere qual è il percorso migliore per arrivare a Lima tra un milione di opzioni per arrivarci ($N=1.000.000$). Per utilizzare i computer per trovare il percorso ottimale dobbiamo digitalizzare 1.000.000 di opzioni, il che implica la loro traduzione in linguaggio bit per il computer classico e in *qubit* per il computer quantistico. Mentre un computer classico

dovrebbe andare uno ad uno analizzando tutti i percorsi fino a trovare quello desiderato, un computer quantistico sfrutta il processo noto come parallelismo quantistico che gli permette di considerare tutti i percorsi contemporaneamente. Ciò implica che, mentre il computer classico ha bisogno dell'ordine di $N/2$ passi o iterazioni, cioè 500.000 tentativi, il computer quantistico troverà il percorso ottimale dopo solo \sqrt{N} operazioni sul registro, cioè 1.000 tentativi.

Nel caso precedente il vantaggio è quadratico, ma in altri casi è addirittura esponenziale, il che significa che con n *qubit* si può ottenere una capacità di calcolo equivalente a 2^n bit. Per esemplificare questo, è comune contare che con circa 270 qubit potremmo avere più stati di base in un computer quantistico - più stringhe di caratteri diversi e simultanei - rispetto al numero di atomi nell'universo, che è stimato intorno ai 280. Un altro esempio è che si stima che con un computer quantistico tra 2000 e 2500 *qubit* potremmo rompere praticamente tutta la crittografia utilizzata oggi (la cosiddetta crittografia a chiave pubblica).

Perché è importante conoscere la tecnologia quantistica?

Siamo in un momento di trasformazione digitale in cui diverse tecnologie emergenti come blockchain, intelligenza artificiale, droni, Internet delle cose, realtà virtuale, 5G, stampanti 3D, robot o veicoli autonomi sono sempre più presenti in molteplici campi e settori. Queste tecnologie, chiamate a migliorare la qualità della vita dell'essere umano accelerando lo sviluppo e generando un impatto sociale, avanzano oggi in modo parallelo. Solo raramente vediamo aziende che sviluppano prodotti che sfruttano combinazioni di due o più di queste tecnologie, come blockchain e IoT o droni e intelligenza artificiale. Sebbene siano destinati a convergere, generando così un impatto esponenzialmente maggiore, la fase iniziale di sviluppo in cui si trovano e la scarsità di sviluppatori e di persone con profili tecnici fanno sì che la convergenza sia ancora un compito in sospeso.

A causa del loro potenziale dirompente, ci si aspetta che le tecnologie quantistiche non solo convergano con tutte queste nuove tecnologie, ma che abbiano un'influenza trasversale su praticamente tutte. Il calcolo quantistico minacerà l'autenticazione, lo scambio e l'archiviazione sicura dei dati, con un impatto importante sulle tecnologie in cui la crittografia ha un ruolo più rilevante, come la sicurezza informatica o la blockchain, e un impatto negativo minore, ma anche da considerare in tecnologie come il 5G, l'IoT o i droni.

Volete praticare il calcolo quantistico?

Decine di simulatori di computer quantistici sono già disponibili in rete con diversi linguaggi di programmazione già in uso come C, C++, Java, Matlab, Maxima, Python o Octave. Inoltre, nuovi linguaggi come il Q#, lanciato da Microsoft. È possibile esplorare e giocare con una macchina quantistica virtuale attraverso piattaforme come IBM e Rigetti.

Mini BlocklyChain è creato dalla società OpenQbit.com che si concentra sullo sviluppo della tecnologia di calcolo quantistico per diversi tipi di settori pubblici e privati.

Perché Mini BlocklyChain è diverso da altri blockchain, semplicemente perché il sistema è stato creato per essere modulare e includere il calcolo quantistico.

- (2) <https://blogs.iadb.org/conocimiento-aberto/es/como-funciona-la-computacion-cuantica/>

29. Allegato "Blocchi estesi per database SQLite".

Per vedere maggiori dettagli sull'uso corretto e tempestivo di ogni blocco che compone l'estensione OpenQbitSQLite controllare l'autore originale dell'estensione nel seguente link.

OpenQbitSQLite è un clone del progetto originale con piccole modifiche da utilizzare con un livello di sicurezza AES.

<https://github.com/frdfsnlght/aix-SQLite>

30. Allegato "Esempio di creazione di un sistema Mini BlocklyChain".

Realizzeremo un sistema di test dove potremo verificare le funzionalità, i vantaggi e la facilità di creazione di un sistema Mini BlocklyChain.

Inizieremo con la progettazione e lo sviluppo del magazzino dove risiederanno le informazioni che abbiamo già definito come una catena di blocchi. Il progetto sarà basato sul database SQLite e, a seconda dell'organizzazione o del progetto, questo può essere facilmente modificato da un altro database come Microsoft SQL Server, MySQL, Maria DB, Oracle, DB2, PostgreSQL, tra gli altri.

Anche se ancora una volta a causa del processo di transazioni e la concomitanza con il database SQLite può essere utilizzato a qualsiasi livello e per l'applicazione aziendale o personale.

La creazione del database chiamato **minibc** in questo avrà una tabella che memorizzerà la stringa del blocco. Questo database sarà incorporato nel codice del programma finale che verrà installato nei nodi, questo luogo di archiviazione è chiamato "asset packaged" è una specie interna in ambienti Blockly come App Inventor.

```
$ sqlite3 minibc.db
SQLite version 3.32.2 2020-06-20 15:25:24
Inserire ".help" per i suggerimenti d'uso.
Collegato ad un database di memoria transitoria.
Utilizzare ".open FILENAME" per riaprire un database persistente.
sqlite> .quit
```

Progettazione dei campi del tavolo **nblock**.

```
CREARE TAVOLA nblock (
    id tasto intero primario AUTOINCREMENTO
    , prevhashVARCHAR NON NULLA
    , newhashVARCHAR NON NULLA
    ntransINTEGER      NON NULLA
    nonceINTEGER       NON NULL
    ,qrng      INTERO NON NULLO
);
;
```

Abbiamo creato la tabella **nblock**.

```
$ sqlite3 minibc.db
SQLite version 3.32.2 2020-06-20 15:25:24
Inserire ".help" per i suggerimenti d'uso.
Collegato ad un database di memoria transitoria.
Utilizzare ".open FILENAME" per riaprire un database persistente.
sqlite> .open minibc.db
sqlite> CREATE TABLE nblock ( id tasto primario intero AUTOINCREMENT , prevhash VARCHAR
NOT NULL , newhash VARCHAR NOT NULL , ntrans INTEGER NOT NULL , nonce INTEGER
NOT NULL ,nonce INTEGER NOT NULL ,qrng INTEGER NOT NULL );
```

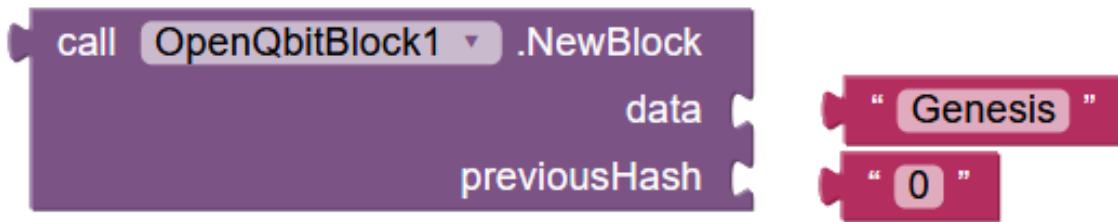
Vedremo qualcosa di simile:

```
$ sqlite3 minibc.db
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
sqlite> CREATE TABLE nblock (
...>     id integer primary key AUTOINCREMENT
...>     , prevhash VARCHAR NOT NULL
...>     , newhash VARCHAR NOT NULL
...>     , ntrans INTEGER NOT NULL
...>     , nonce INTEGER NOT NULL
...>     , qrng INTEGER NOT NULL);
sqlite> .tables
nblock
sqlite> .quit
$
```

Dichiarazione SQL per la creazione della tabella nblock.

Successivamente creeremo l'hash iniziale del sistema chiamato "**Genesis**", che si basa sulla creazione di un nuovo hash del primo blocco della catena di blocchi useremo il blocco (**NewBlock**). Poi creeremo un secondo hash che chiameremo "uno" basato sull'hash "Genesis" perché deve essere coerente con il primo hash perché il test di validazione dell'hash sulla catena di blocchi iniziale deve sempre essere conforme: prevhash = newhash.

NewBlock. Per creare l'hash "Genesis" useremo i parametri di ingresso:



Parametro di ingresso: **dati <Stringa>**, **precedenteHash <Stringa>**

Parametro di uscita: Un SHA256 Hash.

In questo caso useremo come "previousHash" l'iniziale "0" e nel caso di dati di input "data" sarà la parola "Genensis".

Questo ci darà un hash calcolato della combinazione di entrambi i dati:

SHA256 (Genesi0) = eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b6b5adfb09ba54e802e642

La stringa di cui sopra verrà inserita nella tabella nblock, questa stringa deve essere criptata per questo usiamo l'estensione (**OpenQbitEncDecData**).



Useremo l'estensione OpenQbitSSHClient per inserire i dati nel database minibc.db, sarebbe la dichiarazione INSERIRE al database minibc.db della tabella nblock.

```
sqlite3 keystore.db "inserire in nblock (prevhash, newhash, ntrans, nonce, qrng) valori ('0', 'eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b6b5adfb09ba54e802e642', '1', '0', '0', '0')";
```

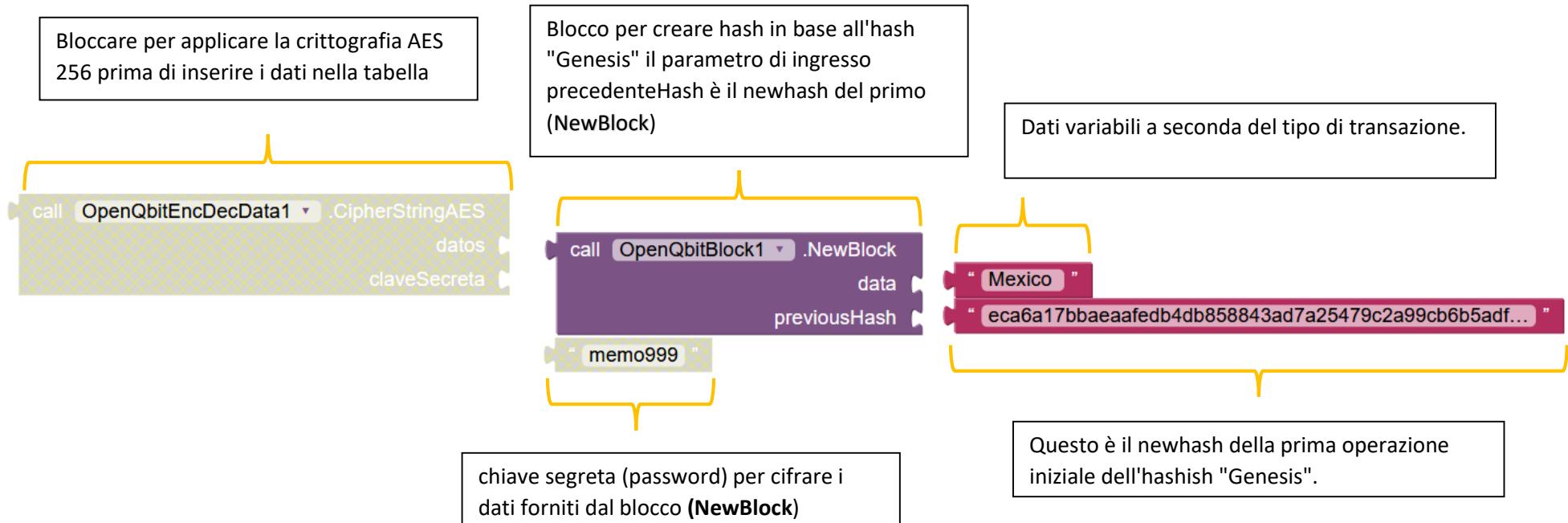
Abbiamo creato l'hash "uno" con lo statement SQL basato sull'hash "Genensis":

```
sqlite3 keystore.db "inserire in nblock (prevhash, newhash, ntrans, nonce, qrng) valori ('eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b6b5adfb09ba54e802e642', f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78ababab59a68', '1', '0', '0', '0');"
```

L'hashish "uno" newhash è calcolato come:

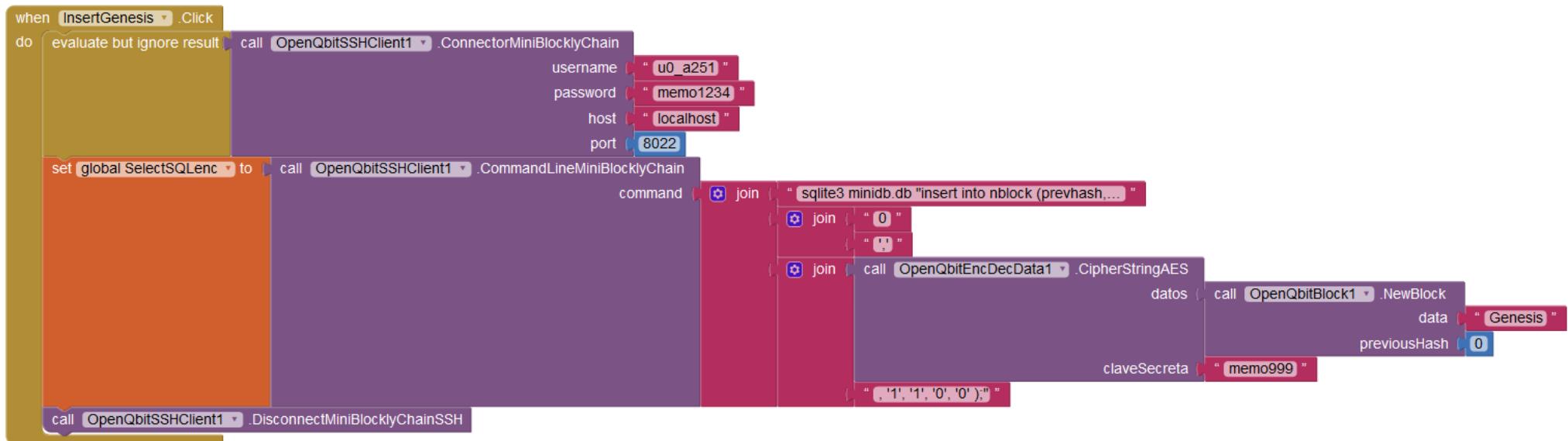
SHA256 (eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b6b5adfbb09ba54e802e642Mexico) =
f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78ab59a68

Secondo hash basato sul primo hash di inizializzazione "Genesis", dobbiamo fare due INSERTI all'inizio perché ogni validazione iniziale della catena di blocchi deve rispettare l'uguaglianza dei campi: prevhash (penultimo dato) = newhash (ultimo dato).



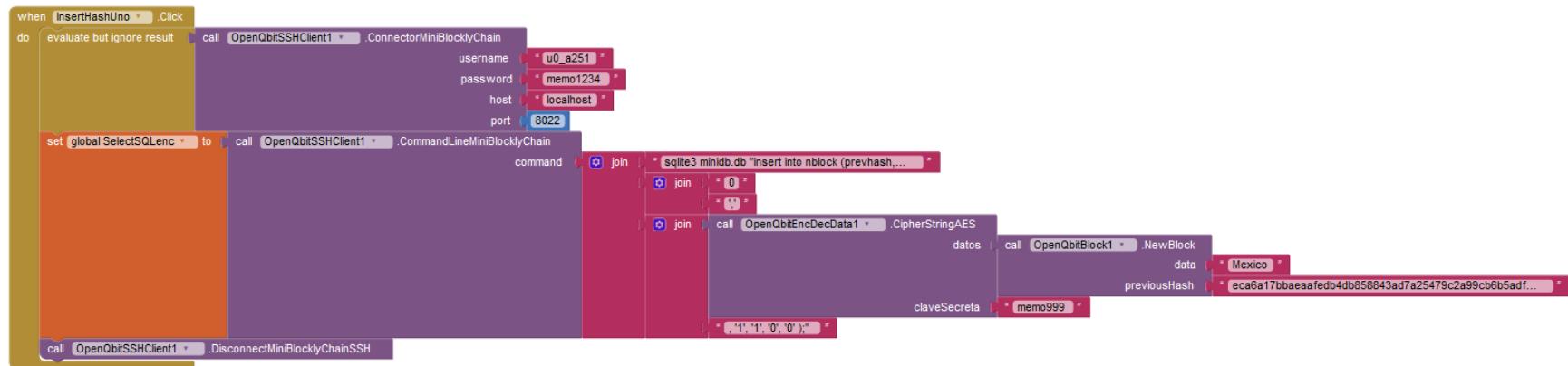
Continuiamo con la creazione dell'esecuzione all'interno di App Inventor degli hash sopra; hash "Genesi" e hash "start".

Ora mostriamo come l'INSERTO dell'hashish "Genesis" sarebbe integrato nella struttura iniziale della catena di blocchi all'interno della tabella nblock nel sistema App Inventor:



Dato che abbiamo inserito i primi dati nella tabella di nblock basati sull'hash "Genesis", procediamo a fare il secondo INSERIMENTO riferito all'hash "uno".

Integrerebbe l'INSERTO dell'hashish "One" nella struttura iniziale della catena di blocchi all'interno della tabella nblock nel sistema App Inventor:



Le due strutture di programmazione precedenti (hash "Genesis" e hash "one") dovrebbero essere integrate dal nodo iniziale del sistema Mini BlocklyChain, un punto importante è che il database minibc.db con tutta la sua struttura interna (tabelle) sarà replicato attraverso la rete Mini BlocklyChain basata su un'architettura "Peer to Peer".

Finora abbiamo completato la struttura di base e fondamentale dello stoccaggio a catena di blocchi ed è pronta a ricevere nuovi blocchi da transazioni future che hanno origine o sono richieste nel sistema.

Abbiamo già inserito i primi dati hash "Genesis" e l'hash "one" nel nostro database **minibc.db**, ora useremo le seguenti istruzioni SQL per interrogare i campi **prevhash** e **newhash** nella tabella **nblock**.

Questo punto sarà fondamentale in quanto sulla base del confronto tra questi due campi sapremo se la catena dei blocchi è coerente e se mantiene l'integrità e la sicurezza delle transazioni. Questo è solo un meccanismo di partenza per rivedere la catena dei blocchi, poiché in futuro rivedremo l'uso del blocco per calcolare l'albero merkle, che sarà un altro strumento essenziale per garantire l'integrità e la sicurezza della catena dei blocchi anche nel nostro sistema.

Per ora rivedremo solo la struttura o le frasi SQL che dovremo utilizzare come abbiamo fatto in precedenza con l'estensione ([OpenQbitSSHClient](#)), con queste potremo consultare i campi prevhash e newhash. Più tardi possiamo fare il semplice confronto dei dati uguali per quel controllo ogni volta che andiamo ad aggiungere un nuovo blocco.

Per prevhash:

```
sqlite3 minibc.db "SELEZIONA prevhash DA nblock ORDINE DALL'ID DESC LIMIT 1;".
```

Per il newhash:

```
sqlite3 minibc.db "SELEZIONA newhash DA nblock ORDINE DALL'ID DESC LIMIT 1;".
```

Ci concentreremo ora su come faremo una richiesta per presentare una nuova negoziazione e/o transazione da inserire nella coda delle transazioni.

Ci sono due fasi come requisiti per inviare una transazione alla coda delle transazioni.

Il primo requisito è che il nostro saldo del conto abbia a disposizione un "patrimonio" sufficiente a coprire l'importo da inviare. Ricordate che i "beni" non possono essere solo un numero o un importo, ma possiamo creare "beni" di qualsiasi tipo a seconda di ogni sistema da creare.

Esempi di attività:

- ✓ Quantità intrinseca di una quantità "virtuale o cripto-valuta" di nuova origine.
- ✓ Dati riferiti a dati digitali (tutti i tipi di documenti)
- ✓ Firme di autenticazione Hash per stringhe o per tutti i tipi di file.
- ✓ Qualsiasi transazione o trasferimento di informazioni digitali o la loro rappresentazione.

Il saldo delle "attività" di ogni nodo si ottiene utilizzando il blocco (**GetBalance**).



Il secondo requisito è quello di fornire i parametri minimi per l'invio di una transazione, che sono

- ✓ Indirizzo della fonte. - è l'indirizzo da cui verranno spediti i beni.
- ✓ Indirizzo di destinazione. - è l'indirizzo dell'utente che riceverà i beni inviati.
- ✓ Attivo. - Sono i dati con valore intrinseco dato in ogni sistema da inviare in ogni transazione.

Gli indirizzi di cui sopra (origine-destinazione) corrispondono alle chiavi pubbliche di ogni utente al momento della creazione dei conti di ogni utente. Ricordate che quando si crea un account utente, si creano due tipi di chiavi pubbliche e private.

La chiave pubblica è l'indirizzo che sarà condiviso in tutto il sistema e che ogni utente del sistema potrà vedere e utilizzare per inviare o ricevere transazioni.

La chiave privata è l'indirizzo che viene utilizzato localmente da ogni nodo e come indica il suo nome è per uso privato che aiuta a creare firme digitali per dare sicurezza alle transazioni inviate, questa chiave non dovrebbe mai essere condivisa ed è per uso locale e unica per il nodo sorgente.

Per poter utilizzare i parametri precedenti ci affideremo a due repository dove sono memorizzati gli indirizzi delle "chiavi private" che si trovano nel database di keystore.db e saranno di uso locale di ogni nodo senza condivisione nel sistema e l'altro repository dove sono memorizzati tutti gli indirizzi delle "chiavi pubbliche" che si trovano nel database di publickeys.db sarà condiviso attraverso il protocollo "Peer to Peer" in tutti i nodi del sistema.

Le banche dati "keystore.db" e "publickeys.db" sono già state create nell'allegato "Creazione di banche dati KeyStore & PublicKeys".

La banca dati e il sistema per la coda delle transazioni sono già stati creati nella sezione "Installazione e configurazione della rete RESTful Mini Sentinel". Dove abbiamo già creato un database per le transazioni chiamato op.sqlite3 in questo abbiamo due tabelle una è "trans" che si occupa delle transazioni (**transaction queue**) e un'altra chiamata "sign" dove sono memorizzate le firme digitali di ogni operazione.

Il sistema SQLite RESTful è la rete di backup in questa occasione lo utilizzeremo per comodità e per testare il sistema di backup, tuttavia, per modificare e utilizzare lo stesso database op.sqlite3 in un modello "Peer to Peer" dovremo utilizzare il database solo in un modello condiviso di nuovo nel sistema di sincronizzazione, questo lo vedremo più avanti.

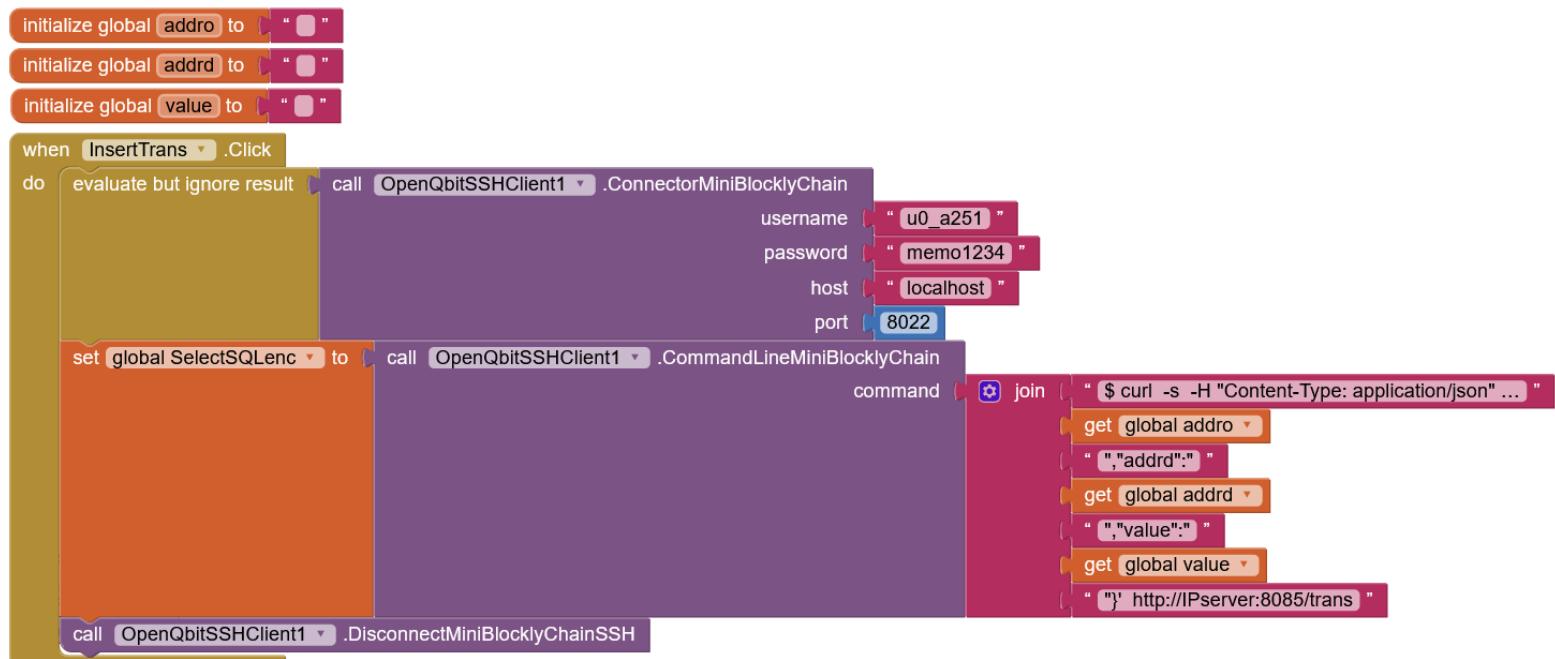
Inizieremo ad inviare le operazioni alla coda delle transazioni utilizzando RESTful applicato al database dell'op.sqlite.

Abbiamo creato una nuova transazione nella tabella "trans"

```
$ curl -s -H "Content-Type: application/json" -d
'{"addr": "MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9",
"addrd": "MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh", "value": "999"}'
http://IPserver:8085/trans
```

```
# uscite
{
  "id": 1,
  "addr": "MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9",
  "addrd": "MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh",
  "valore": "999"
}
```

Implementazione in App Inventor:



I parametri di input: addro, addrd, valore sono variabili che possono essere inserite nell'algoritmo e che vengono estratte dal database di keystore.db

Vedi Appendice "Creazione di banche dati KeyStore".

Ora inseriamo le firme digitali della transazione precedente. Nella tabella "**segno**"

```
$ curl -s -H "Content-Type: application/json" -d '{  
  "trans_id":1  
  "segno": "59ahGVn8pZ4oduxtTWCCvRiqjWEEy1c62",  
  "hash": "f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78abab59a68"}"  
http://IPserver:8085/sign  
  
# uscite  
{  
  "id": 1,  
  "trans_id": 1,  
  "segno": "59ahGVn8pZ4oduxtTWCCvRiqjWEEy1c62",  
  "hash": "f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78ab59a68".  
}
```

NOTA: La firma digitale (segno) deve essere ottenuta prima di inviare la dichiarazione del comando curl, viene generata con il blocco (**GenerateSignature**).

I parametri di input: Trans_id, sign, hash sono variabili che possono essere inserite nell'algoritmo e la firma digitale della transazione sarà generata con il blocco (**GenerateSignature**).

Vedremo ora la procedura per la generazione di una firma digitale da applicare ad ogni transazione che viene effettuata.

Generazione di firma digitale per la transazione. Quando usiamo il blocco (**GenerateSignature**) avremo come risultato un **file** di tipo **.sig** questo file lo useremo per essere salvato nel campo sign nella tabella "**sign**", questa firma sarà usata quando la coda delle transazioni viene elaborata ed è un altro parametro per dare sicurezza tra l'origine e la destinazione, così come l'importo o il tipo di transazione tra di loro.

Per gestire dati binari come il file.sig dobbiamo convertirli in base64 e poi memorizzarli nella variabile segno nella tabella "sign". Per fare questo useremo il blocco (**EncoderFileBase64**).

Come viene calcolato il valore del campo hash per la tabella "segno" di ogni transazione:

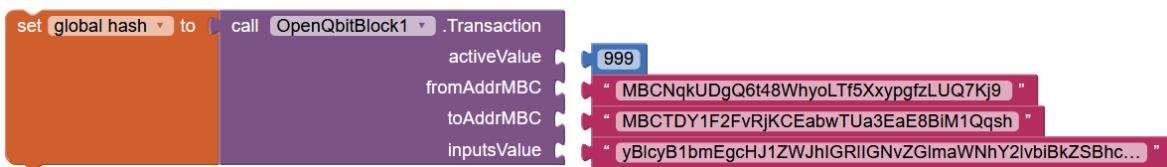
Transaction Hash = SHA256(Indirizzo di origine + Indirizzo di destinazione + Asset + fileBase64.sig)

Esempio di hash tipo SHA256:

SHA256(MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9 + MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh + 999 +).

Uscita: 9d45198faaef624f2e7d1897dd9b3cede6ecca7fbac516ed1756b350fe1d56b4

Per il calcolo dell'hashish dovremo appoggiarci al blocco (**transazione**).



Il parametro di uscita è l'hash che verrà memorizzato per ogni transazione che viene inviata da qualsiasi nodo del sistema. Questo viene memorizzato nel campo segno della tabella "segno" nella base op.sqlite

Con l'operazione precedente abbiamo fatto in modo che solo un hashish unico e irripetibile rappresenti ogni transazione inviata alla coda e questo hash rappresentativo è la totalità delle informazioni trasmesse.

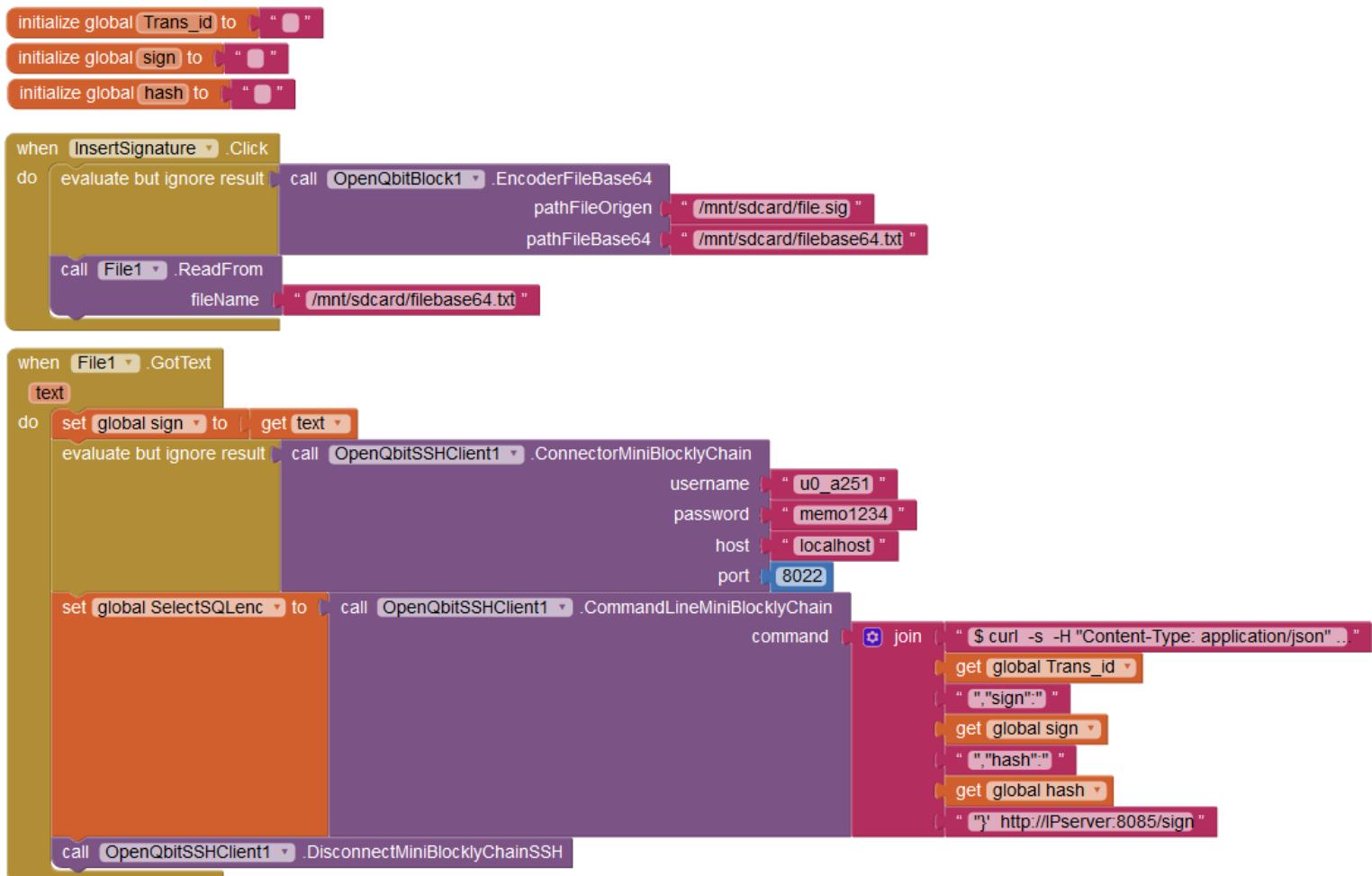
L'unica cosa che manca è includere la variabile Trans_id, che può essere un identificatore per differenziare quale nodo viene inviato per ogni transazione. Nel nostro esempio metteremo la variabile Trans_id con un valore fisso di "1". Perché è il primo nodo che viene configurato nella nostra rete. Questo valore può variare la sintassi in base ad ogni particolare disegno, quindi per semplicità abbiamo scelto un semplice identificatore.

Poiché abbiamo tutte le variabili definite creeremo la struttura e la sequenza di esecuzione dei blocchi all'interno di App Inventor, per eseguire l'INSERTO nella tabella "segno".

NOTA: È importante tenere conto che ogni invio di una transazione è composto da due INSERTI nel database dell'op.sqlite3, uno deve essere fatto nella tabella "trans" e i rispettivi valori nella tabella "segno".

Nella progettazione del database op.sqlite3 sono state create due tabelle separate, in quanto in futuro è possibile criptare la tabella "segno" solo perché contiene informazioni che non devono essere inviate in rete in modo piatto, questa opzione è lasciata alla considerazione di ogni progetto futuro.

Creeremo la struttura di esecuzione dei blocchi e dei metodi all'interno dell'App Inventor dell'INSERT nella tabella "segno".



Fino a questo momento abbiamo già terminato l'INSERIMENTO alla tabella "**segno**" e alla tabella "**trans**", che si trovano all'interno del database dell'**op.sqlite3** e i dati inseriti in queste due tabelle verranno utilizzati per creare la coda delle transazioni che verranno inviate a tutti i nodi per la loro elaborazione.

In questo momento useremo il connettore Java SQLite-Redis Connector, che eseguirà la conversione dei dati dal database op.sqlite3 al database Redis. Vedere l'appendice "Java SQLite-Redis Code Connector".

Dopo aver implementato il Connnettore SQLite-Redis, la coda delle transazioni sarà consegnata a tutti i nodi del sistema attraverso il sistema Redis.

Inizieremo il processo di come possiamo ricevere la coda delle transazioni in modo adeguato per poterla elaborare.

La coda delle transazioni sarà consegnata attraverso il servizio Redis, un database in tempo reale che ha i suoi rispettivi blocchi di controllo in ambiente App Inventor.

Configurazione dell'ambiente Redis all'interno del sistema Blockly di App Inventor.

Un punto importante da notare è che i blocchi per il controllo di un server Redis fino al momento della scrittura di questo manuale sono disponibili solo nel sistema App Inventor. In caso di utilizzo di un sistema Blockly diverso da App Inventor si dovrà utilizzare l'esecuzione di Redis CLI (Command-Line) attraverso l'invio di comandi al terminale Termux tramite l'estensione ([OpenQbitSSHClient](#)).

Esempio di utilizzo in Redis CLI (riga di comando):

Per ottenere tutte le **etichette** o le chiavi di Redis.

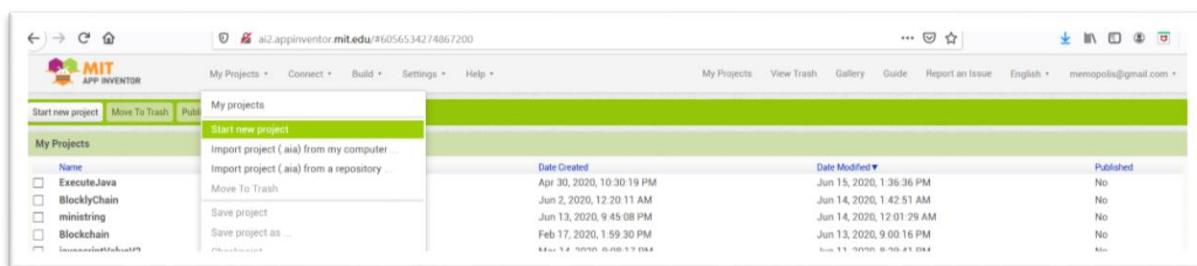
\$ redis-cli TASTI '*'.

Per ottenere valore da una chiave.

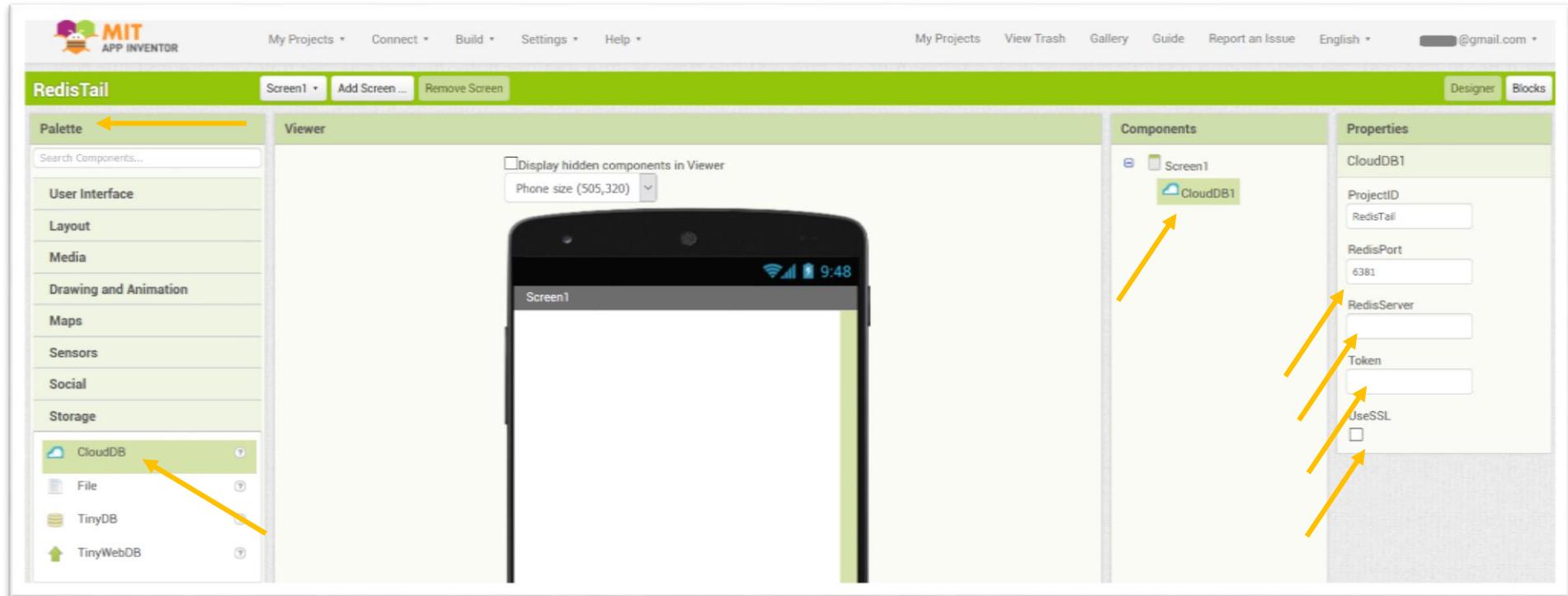
\$ redis-cli GET <chiave>

Nel nostro caso, poiché stiamo usando l'inventore di App, esamineremo come utilizzare i blocchi per lavorare con Redis.

All'interno di App Inventor abbiamo diversi blocchi da utilizzare con Redis, abbiamo iniziato a creare un nuovo progetto che realizzeremo: I miei progetti > Inizia un nuovo progetto



Dopo aver creato il progetto andiamo in alto a sinistra e nella sezione "Palette" clicchiamo su "Storage" e trasciniamo l'oggetto "CloudDB" questo integrerà tutte le funzionalità per configurare una connessione ad un server Redis.



La configurazione è molto semplice, basta dare i seguenti parametri per potersi collegare al nostro server Redis (Locale) che è in esecuzione nel nostro nodo (Cellulare).

ProjectID. - Questo è l'ID che farà partire l'etichetta che identifica la coda delle transazioni in Redis.

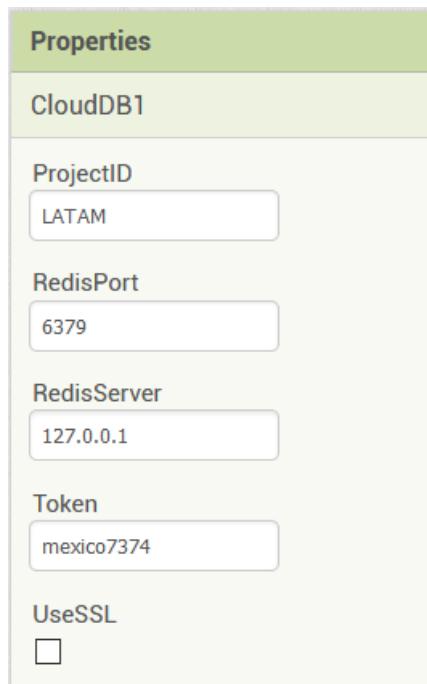
RedisPort. - Questa è la porta del server Redis dove ci connetteremo di default è 6379

RedisServer. - Questo è il dominio o IP locale del nodo nel nostro caso e quello di tutti i nodi sarà 127.0.0.1

Gettone. - Questa è la password del server Redis abilitato alla connessione.

UtilizzareSSL. - Questa opzione ci dà l'opportunità di utilizzare i certificati SSL nel nostro caso lo lasciamo non abilitato.

Nel nostro modello e nella progettazione del sistema avremo i seguenti parametri in tutti i nodi della rete.



È giunto il momento di rivedere i blocchi che ci forniscono il controllo e l'elaborazione dei dati in un ambiente di database Redis.

Iniziamo con il blocco del metodo (**DataChanged**)



Questo metodo ci darà due valori ogni volta che ci sarà un cambiamento nel server Redis che abbiamo configurato:

tag. - Questo valore è il tag che identifica la coda delle transazioni.

valore. - Questo valore contiene l'elenco di tutte le transazioni inviate per l'elaborazione. Questo valore è una lista di stringhe di tipo <Stringa>.

Nel caso del "valore" è dove inizieremo ad eseguire il nostro trattamento delle informazioni come segue.

Poiché ci fornite una catena di tutte le transazioni di valore dell'hashish, inizieremo controllando se questa catena è valida e verificheremo se non è stata modificata dalla sua origine. Lo facciamo utilizzando un algoritmo molto utile per la verifica di grandi quantità di informazioni.

Useremo l'algoritmo ad albero di Merkle. L'applicazione di questo algoritmo ci dà una sicurezza nell'integrità delle informazioni che riceviamo (coda di transazione).

Vediamo il seguente esempio di coda di transazione in Redis, eseguiamo la Redis CLI (Command-Line) da un terminale Termux per controllare la chiave "LATAM:HASH", dobbiamo aver già eseguito il connettore SQLite-Redis per poter consultare questa chiave.

```
C:memor>redis-cli  
127.0.0.1:6379> ottenere LATAM:HASH  
"9d45198faaef624f2e7d1897dd9b3cde6ecca7fbac516ed1756b350fe1d56b4,"  
f71c801a5fd25fd25fc303ebc8c616204b4877ffb93006ec6a88bc30acf43ec250f5,"\60f8a3bc  
ac1ea7d38e86efbc3e3e3e00480807d23f980391000766e804ed14ecb2 ,  
8a6dfe1d38c22e0f9212052efa6136da3edf1fb1bb2a3e25224ac3d689124b754]  
127.0.0.1:6379>
```

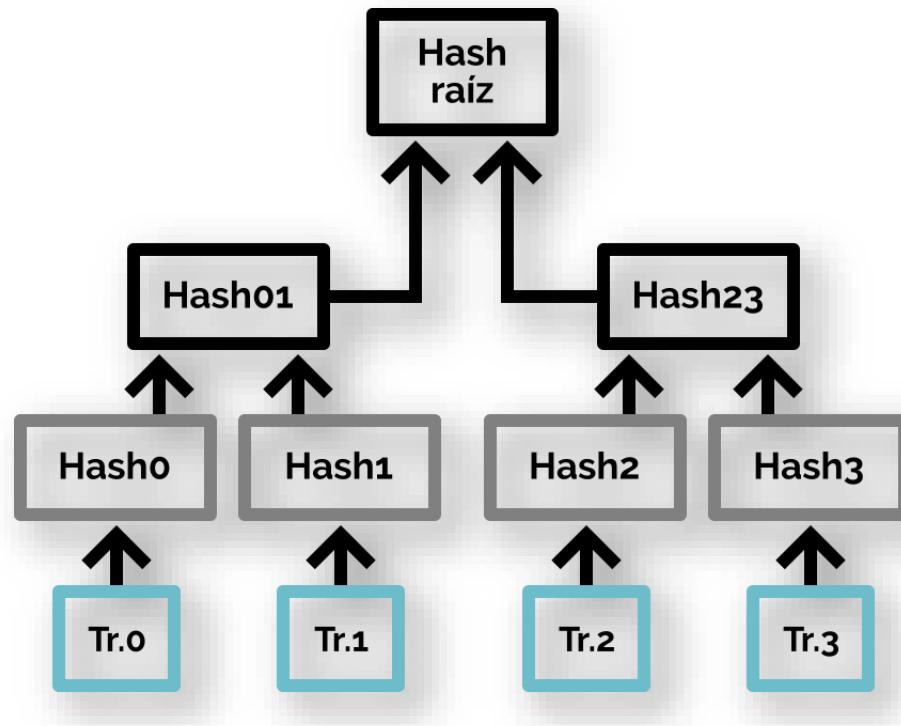
La precedente coda di transazioni ha solo tre elementi, poiché vediamo quattro hash che rappresentano una singola transazione che la compongono e sono gli hash di ogni transazione che è stata inizialmente iniettata nel database dell'op.sqlite3 all'interno delle tabelle "trans" e "sign".

Ora è il momento di capire come funziona l'albero merkle.

Un hash merkle tree è una struttura ad albero di dati binari o non binari in cui ogni nodo che non è un foglio è etichettato con l'hash della concatenazione delle etichette o dei valori dei suoi nodi figli. Sono una generalizzazione delle liste di hash e delle stringhe di hash.

Nel nostro caso faremo il seguente calcolo per calcolare l'albero merkle per quattro elementi questo viene fatto calcolando il risultato di prendere coppie di dati concatenati e ottenere i rispettivi hash, i risultati del primo livello verranno applicati ai risultati del secondo livello fino a quando non ci sarà un solo elemento finale, questo sarà chiamato l'hash merkleroot (root hash).

Vediamo il seguente diagramma che descrive questo processo.



La coda delle transazioni basate sull'hash verrà estratta tramite il blocco (**GetMerkleRoot**)



Radice di hash:
51431822de7c94b90dc06d47b8f6275f315a4976c8479d30c32747fa90325432

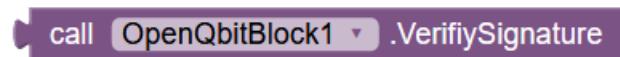
Il risultato viene poi confrontato con la chiave LATAM:merkleroot del sistema locale Redis ed entrambe le chiavi devono corrispondere per verificare l'integrità dei dati.

Un altro punto di sicurezza fornito dall'albero merkle è la conferma che una specifica transazione è inclusa nella coda di transazione dalla sua origine e che non è stata introdotta in modo fraudolento da alcun mezzo di comunicazione esterno o interno.

Nel caso in cui la catena sia di elementi dispari nella sua sommatoria, l'ultimo elemento si duplica per avere una disposizione e iniziare l'esecuzione dell'algoritmo.

Un altro punto non meno importante è il momento di convalidare che ogni transazione corrisponda alla sua origine-destinazione e che il bene sia quello inviato dall'indirizzo di origine.

Qui si trova il blocco (VerifySignature).



Prima di eseguire il blocco precedente, è necessario scaricare il file (file.sig) in formato binario dalla tabella "segno":

```
sqlite3 op.sqlite3 "seleziona segno da segno dove=id_addr;"
```

id_addr: Questo è l'id dell'indirizzo sorgente della tabella "trans".

La precedente richiesta di ricerca ci fornirà i dati in formato Base64 della firma digitale della transazione corrente, questo per convertirla nel suo formato originale (binario) avremo bisogno del Block(**DecoderFileBase64**).

Poiché abbiamo il nostro file binario (file.sig) dovremo caricare nel sistema la chiave pubblica dell'origine e la chiave pubblica del destinatario anche in formato binario, avendo i quattro dati nei rispettivi formati sarà il momento di eseguire la verifica della firma digitale nel sistema.

- ✓ Indirizzo di casa a chiave pubblica
- ✓ Indirizzo a chiave pubblica del destinatario
- ✓ Attivo inviato.
- ✓ Firma digitale (file.sig)

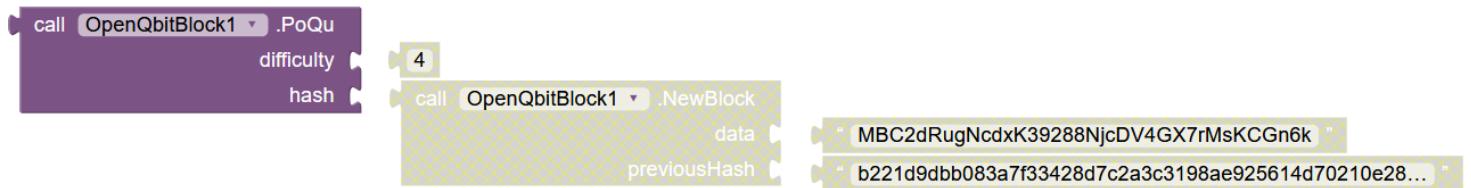
Le chiavi pubbliche in formato binario possono essere scaricate nel formato appropriato (binario) dal database condiviso publickeys.db

Questo processo deve essere eseguito per ogni singola operazione nella coda delle transazioni.

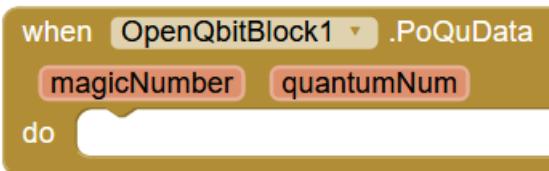
Infine esamineremo come il sistema sceglie un nodo in modo consensuale per poter essere scelto per aggiungere il blocco successivo alla catena di blocchi ed essere quello che elabora la coda delle transazioni.

Questo modo di scegliere il nodo vincente per elaborare la coda delle transazioni si basa sul nostro algoritmo sviluppato per un sistema Mini BlocklyChain.

Come abbiamo implementato il consenso PoQu "Proof of Quantum". Questo processo di consenso si basa sulla generazione di numeri casuali quantistici e viene applicato utilizzando il blocco (**PoQu**).



Per prima cosa otteniamo due parametri che il blocco (**PoQu**) ci fornisce nel metodo (**PoQuData**) i parametri sono il **magicNumber** e il **quantumNum**.



Il **magicNumber** è il numero "nonce", è un numero intero che si ottiene facendo un PoW interno con difficoltà non superiore a 5, questo numero ha il compito di dare un primo requisito al nodo con il **magicNumber** il nodo sarà in grado di fare un requisito per ottenere un numero del sistema di generazione di numeri quantistici casuali che si trova nell'intervallo da 0 a 1, questo numero darà una probabilità stabilita casualmente per il nodo in esecuzione che sarà memorizzata nella tabella chiamata "voto".

La tabella "voto" memorizzerà il **quantumNum** calcolato da ogni nodo, questa memorizzazione sarà fatta con un certo numero di nodi fino ad un certo tempo stabilito in ogni progetto di sistema si raccomanda di avere delle voci $((N/2) + 1)$ dove "N" è la quantità di nodi disponibili nel sistema e può essere stabilita o controllata da un'azione nello strumento di gestione dei compiti "cron" di ogni nodo.

Un punto importante è che a questo punto si dovrebbe già aver stabilito una sincronizzazione temporale locale di ogni nodo attraverso il sistema automatico del cellulare in caso di utilizzo della rete "**Peer to Peer**" nella trasmissione della coda di transazione.

La configurazione dell'agente di cron nei nodi. Vedere il paragrafo "Sincronizzazione temporale nei nodi del sistema (telefono cellulare) in minuti e secondi".

In questo esempio, dato che stiamo usando la rete di comunicazione di backup, non abbiamo bisogno della sincronizzazione di minuti e secondi per i nodi perché il nostro esempio occupa uno schema "**Client-Server**", questo tipo di comunicazione è solo nel processo di trasmissione della coda delle transazioni. Tutti gli altri processi tra i nodi passano attraverso una comunicazione "**Peer to Peer**".

Ora esamineremo la struttura, la progettazione e la creazione della tabella di "voto" che si troverà all'interno del database chiamato "quorum.db" che creeremo anche in questo esempio.

\$ sqlite3

SQLite versione 3.32.2 2020-06-20 15:25:24

Inserire ".help" per i suggerimenti d'uso.

Collegato ad un database di memoria transitoria.

Utilizzare ".open FILENAME" per riaprire un database persistente.

sqlite> .open quorum.db

sqlite> CREATE TABLE vote (id integer chiave primaria AUTOINCREMENT NOT NULL, node_imei VARCHAR NOT NULL, quantumNum INTEGER NOT NULL, nonce INTEGER NOT NULL);

sqlite> .quit

La creazione della stessa tabella di **voto** è mostrata qui sotto, ma è l'introduzione della dichiarazione SQL in forma segmentata:

\$ sqlite3

SQLite versione 3.32.2 2020-06-20 15:25:24

Inserire ".help" per i suggerimenti d'uso.

Collegato ad un database di memoria transitoria.

Utilizzare ".open FILENAME" per riaprire un database persistente.

sqlite> .open quorum.db

sqlite> CREARE TAVOLA voto (

...> id tasto primario intero AUTOINCREMENTO

...> nodo_imei VARCHAR NON NULL,

...> quantumNum INTEGER NOT NULL,

...> nonce INTEGRO NON NULLA

...>);

sqlite> .tavoli

voto

sqlite> .quit

Vedremo qualcosa di simile nella creazione del "quorum.db" di base e nel voto a tavolino.

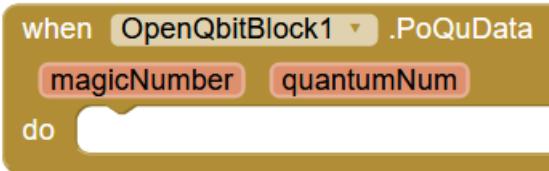
```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open quorum.db
sqlite> CREATE TABLE vote (
...>     id INTEGER primary key AUTOINCREMENT,
...>     node_imei VARCHAR NOT NULL,
...>     quantumNum INTEGER NOT NULL,
...>     time INTEGER NOT NULL
...> );
sqlite> .tables
vote
sqlite> .quit
$
```

Vediamo ora come otteniamo i valori dalla tabella del "voto".

nodo_imei. - **Otteniamo** questo valore utilizzando il blocco (**GetDeviceID**).

```
call [OpenQbitBlock1] .GetDeviceID
```

quantumNum - Questo valore è uno dei risultati del metodo (**PoQuData**) che si ottiene utilizzando il blocco (**PoQu**).



nonce. - Questo valore si ottiene dall'aver già eseguito il blocco (**PoQu**) in modo integrale e uguale al magicNumber del metodo (**PoQuData**).

Dopo aver completato gli INSERTI nella tabella "votazione", è necessario fare una copia della banca dati.

Dopo un certo tempo e in base alla progettazione di ogni sistema, il servizio "cron" verrà eseguito su ogni nodo di rete e avrà il successivo SELECT da elaborare nella tabella "voto":

```
SELECT node_imei DA votare DOVE magicNumber= (SELECT max(magicNumber) DA votare);
```

Il precedente SELECT restituisce il risultato IMEI con maggiore probabilità, ora ogni nodo che esegue SELECT farà un confronto del suo IMEI con il risultato IMEI e solo il nodo che corrisponde creerà un file con il numero di probabilità più alto che verrà replicato nella rete "Peer to Peer" da un file con il formato IMEI.mbc che conterrà l'IMEI del nodo vincente.

Il nodo vincente sarà in grado di avviare l'elaborazione della coda delle transazioni. Sulla base di tutti i blocchi di cui sopra.

Due punti importanti sono che, a seconda della creazione di ogni sistema, tre processi dovranno essere rivisti e personalizzati da ogni progettista.

1.- Quando il nodo vincente inizia ad elaborare la coda delle transazioni, deve essere implementato un metodo o un processo in cui si deve verificare che il nodo vincente sia online e che la comunicazione con la rete non sia andata persa.

2.- Quando inizia l'elaborazione della coda delle transazioni, il nodo vincente deve avviare due flag di controllo che indicano l'inizio dell'elaborazione e un altro per confermare che l'elaborazione della coda delle transazioni è stata completata. Questi due flag devono essere condivisi in rete con tutti i nodi, questo aiuterà a localizzare errori di connessione o errori di elaborazione o troppo tempo di elaborazione.

3.- In caso di fallimento della comunicazione o di altro evento in cui il nodo vincente non è stato in grado di elaborare la coda della transazione, deve essere scelto il nodo successivo nell'intervallo di probabilità immediato.

Il punto precedente può essere controllato con un servizio che verifica che il nodo vincente sia online e possa utilizzare il servizio "cron", lo script deve essere sviluppato per ogni caso progettato, tuttavia, di seguito viene mostrato un esempio generico di script di shell in modo che possa essere modificato secondo le esigenze di ogni sistema Mini Blocklychain.

```
#!/bin/bash

dir="/data/data/com.termux/files/home/Sync/imei";
se [ !"$(ls $directory)" ]
poi
sqlite3 quorum.db "UPDATE vote SET magicNumber=0 WHERE magicNumber=
(SELEZIONA max(magicNumber) DALLA VOTA);".

altro

MAX_NUM=$(sqlite3 quorum.db "SELEZIONA max(magicNumber) DAL voto;")
IMEI_quorum=$(sqlite3 quorum.db "SELECT node_id FROM vote WHERE=MAX_NUM")
IMEI_local=$(cat device_imei) // Usa il blocco (GetDevice)
se [ IMEI_quorum -eq IMEI_local ]
poi
toccare $MAX_NUM > IMEI.mbc
fi
fi
uscita
```

31. Allegato "Integrazione con gli ambienti Ethereum & Bitcoin".

Ora vedremo come integrare i due sistemi a catena di blocchi più noti a livello mondiale specializzati in crittometrie come Ethereum e Bitcoin.

Cominciamo con l'installazione del software che ci aiuterà ad eseguire tutte le possibili transazioni nell'ambiente Ethereum.

Che cos'è l'Ethereum?

Ethereum è una piattaforma open source, decentralizzata a differenza di altre catene a blocchi, Ethereum può fare molto di più. È programmabile, il che significa che gli sviluppatori possono utilizzarlo per creare nuovi tipi di applicazioni.

Queste applicazioni decentralizzate (o "dapps") ottengono i vantaggi della tecnologia di crittomontaggio e della catena a blocchi. Sono affidabili e prevedibili, il che significa che una volta "caricati" nell'Ethereum, funzioneranno sempre secondo i tempi previsti. Possono controllare gli asset digitali per creare nuovi tipi di applicazioni finanziarie. Possono essere decentralizzati, il che significa che nessuna entità o persona li controlla.

In questo momento, migliaia di sviluppatori in tutto il mondo stanno creando applicazioni su Ethereum e inventando nuovi tipi di applicazioni, molte delle quali possono essere utilizzate oggi:

- Portafogli di cripto-valuta che consentono di effettuare pagamenti istantanei a basso costo con ETH o altri asset
- Applicazioni finanziarie che consentono di prendere in prestito, prestare o investire i vostri beni digitali
- Mercati decentralizzati, che consentono lo scambio di beni digitali, o anche lo scambio di "previsioni" sugli eventi del mondo reale.
- Giochi in cui si ha un patrimonio nel gioco e si possono anche vincere soldi veri.
- Ha contratti intelligenti che sono programmi con accordi da eseguire quando i locali con cui è stato elaborato o creato sono soddisfatti.

I contratti intelligenti condividono le analogie con le DApp (applicazioni decentralizzate), ma le separano anche da alcune importanti differenze.

Come i contratti intelligenti, una DApp è un'interfaccia che collega un utente a un servizio di un provider attraverso una rete peer network decentralizzata. Ma, mentre i contratti intelligenti richiedono la creazione di un numero fisso di partecipanti, le DApp non hanno alcun limite al numero di utenti. Inoltre, non si limitano solo ad applicazioni finanziarie come i contratti intelligenti: una DApp può servire a qualsiasi scopo si possa pensare.

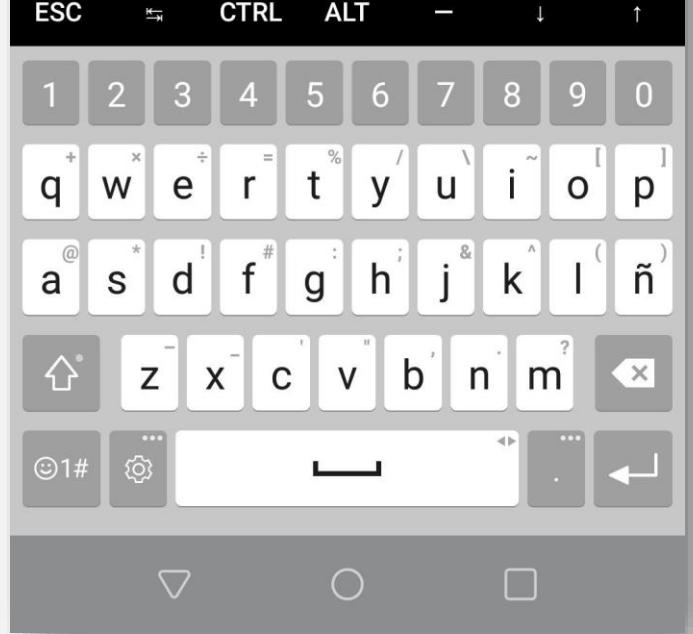
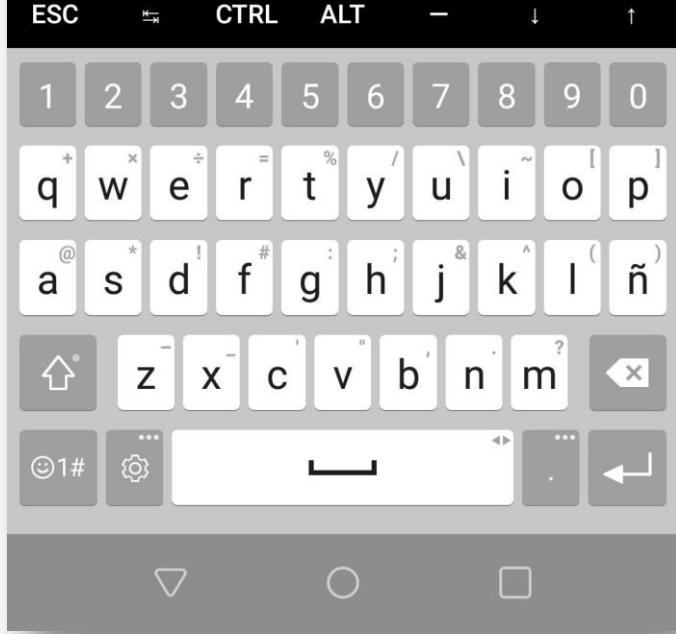
Nel nostro caso useremo la libreria chiamata "Web3j" che è sviluppata in Java, e che permette di interagire con la catena di blocchi dell'Ethereum in modo semplice ed intuitivo.

Eseguire il seguente comando per installare "Web3j":

```
$ curl -L get.web3j.io | sh
```

```
#####
##### 100.0%
Installing Web3j...
Removing downloaded archive...

Web3j was successfully installed.
To use web3j in your current shell run:
source $HOME/.web3j/source.sh
When you open a new shell this will be performed
automatically.
To see what web3j's CLI can do you can check the
documentation bellow.
https://docs.web3j.io/command\_line\_tools/
```



Poi dobbiamo creare la variabile d'ambiente **\$JAVA_HOME** con il percorso dove si trova l'eseguibile "java" poiché la libreria cercherà questa variabile per poter essere eseguita con successo.

```
$ JAVA_HOME= /data/data/com.termux/files/usr/bin
```

Una volta creata la variabile dobbiamo andare nella directory dove è stata installata la libreria "Web3j" eseguendo il seguente comando, un punto importante è che la directory è nascosta dopo aver dr il comando di "cd" mettiamo un punto "." E poi la directory senza spazi, come segue:

```
$ cd .web3j
```

Una volta dentro si verifica se la libreria funziona correttamente con il seguente comando:

```
./web3j versione
```

Il risultato è qualcosa di molto simile a:

```

$ ls
source.sh  web3j  web3j-4.5.16
$ ./web3j version

Version: 4.5.16
Build timestamp: 2020-03-06 14:13:49.943 UTC
$ 
```

Più tardi creeremo un Portafoglio da utilizzare nell'ambiente blockchain dell'Ethereum nel seguente modo:

\$./web3j portafoglio creare

Il comando precedente ci dà l'indirizzo di ethereum:

4598fe2fd6afe2508f58343c7d42f2ab492edf34

```
$ ./web3j wallet create

Please enter a wallet file password:
Please re-enter the password:
Please enter a destination directory location [/data/data/com.termux/files/home/.ethereum/testnet/keystore]:
Wallet file UTC--2020-06-27T06-12-23.819752000Z-4598fe2fd6afe2508f58343c7d42f2ab492edf34.json successfully created in: /data/data/com.termux/files/home/.ethereum/testnet/keystore
$
```

Ne risulta un file in formato JSON contenente l'indirizzo e i dati criptati da utilizzare successivamente per generare la chiave privata del conto appena creato.

Questo file di tipo JSON verrà creato in una directory predefinita chiamata keystore.

Da qui in poi possiamo già eseguire operazioni utilizzando e/o eseguendo il comando Web3j.

Possiamo farlo utilizzando l'estensione (**ConnectorSSHClient**).

Per sapere come utilizzare i diversi parametri e le operazioni della libreria "Web3j" possiamo aiutarci consultando la documentazione sul suo sito ufficiale.

<https://docs.web3j.io/>

Per l'ambiente Bitcoin, abbiamo due opzioni: usare il blocco () che genera l'account Bitcoin e le sue rispettive chiavi

pubbliche e private. Possiamo usare queste chiavi per integrarsi nell'ambiente Bitcoin installando la libreria java "Bitcoij".

\$ npm installare bitcoinj



```
$ npm install bitcoinj
+ bitcoinj@0.0.0
added 1 package from 1 contributor and audited 2
09 packages in 20.528s
$
```

Per poter utilizzare questa biblioteca ci affideremo al suo sito ufficiale.

<https://bitcoinj.github.io/>

Una seconda opzione da integrare nell'ambiente Bitcoin blockchain è quella di installare il pacchetto Bitcoind per Termux come mostrato di seguito.

\$ apt installare bitcoin



```
$ pkg install bitcoin
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  bitcoin
0 upgraded, 1 newly installed, 0 to remove and 1
9 not upgraded.
Need to get 3601 kB of archives.
After this operation, 15.0 MB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-packa
ges-24 stable/main arm bitcoin arm 0.20.0 [3601
kB]
Fetched 3601 kB in 2s (1253 kB/s)
Selecting previously unselected package bitcoin.
(Reading database ... 19110 files and directorie
s currently installed.)
Preparing to unpack .../bitcoin_0.20.0_arm.deb .
..
Unpacking bitcoin (0.20.0) ...
Setting up bitcoin (0.20.0) ...
$
```

Ora, quando eseguiamo l'agente bitcoind sul terminale Termux, questo creerà automaticamente un indirizzo nella directory:

/data/dati/dati/com.termux/files/hme/.bitcoin

In questo caso di Bitcoin ci affideremo alla seguente documentazione dell'agente "Bitcoind".

In questo caso possiamo anche utilizzare l'estensione (**ConnectorSSHClient**) per eseguire l'agente "bitcoind" a seconda delle esigenze.

Descrizione dei parametri per l'utilizzo con bitcoind.

NOME

bitcoind - lanciare Bitcoin Core Daemon

SINOSI

bitcoind [opzioni] Avvio *Bitcoin Core Daemon*

DESCRIZIONE

Avviare Bitcoin Core Daemon

OPZIONI

-?

Stampare questo messaggio di aiuto e uscire

-alertnotify=<cmd>

Eseguire il comando quando si riceve un allarme rilevante o si vede una forcella molto lunga (%s in cmd viene sostituito dal messaggio)

-assumevalid=<esempio>

Se questo blocco è nella stringa si assume che lui e i suoi antenati siano validi e potenzialmente salta la verifica di scrittura (0 per verificare tutti, default:

00000000000000000000f1c54590ee18d15ec70e68c8cd4cfbadb1b4f11697eee, testnet:
0000000000000000000037a8cd3e06cd5edbfe9dd1dbcc5dacab279376ef7cfxfc2b4c75)

-blocknotify=<cmd>

Eseguire il comando quando cambia il blocco migliore (%s in cmd viene sostituito dall'hash del blocco)

-blockreconstructionextratxn=<n>

Transazioni extra da tenere in memoria per le ricostruzioni di blocchi compatti (default: 100)

-blocksdir=<dir>

Specificare la directory dei blocchi (default: <datadir>/blocchi)

-solo in blocco

Se rifiutate le transazioni dei partner della rete. Le transazioni di portafoglio o RPC non sono interessate. (predefinito: 0)

-conf=<archivio>

Specificare il file di configurazione. I relativi percorsi saranno preceduti dalla posizione del datadir. (predefinito: bitcoin.conf)

-daemon

Corre sullo sfondo come un demone e accetta i comandi

-datadir=<dir>

Specificare la directory dei dati

-dbcache=<n>

Dimensione massima della cache del database <n> MiB (da 4 a 16384, default: 450). Inoltre, la memoria mempool inutilizzata è condivisa per questa cache (vedi -maxmempool).

-debuglogfile=<file>

Specificare la posizione del file di log del debug. I percorsi relativi saranno preceduti da un prefisso con una posizione di datadir specifica per la rete. (-nodebuglogfile da disabilitare; default: debug.log)

-includeconf=<file>

Specificare un file di configurazione aggiuntivo, relativo al percorso **-datadir** (può essere utilizzato solo dal file di configurazione, non dalla riga di comando)

-loadblock=<file>

Importare i blocchi dal file esterno blk000???.dat all'inizio

-maxmempool=<n>

Mantenere il pool di memoria delle transazioni al di sotto di <n> megabyte (predefinito: 300)

-maxorphantx=<n>

Mantenere in memoria il massimo <n> di transazioni scollegabili (default: 100)

-mempoolexpiry=<n>

Non tenere le transazioni in mempool per più di <n> ore (default: 336)

-par=<n>

Imposta il numero di fili di verifica nel cruscotto (da -6 a 16, 0 = auto, <0 = lascia tutti i core liberi, default: 0)

-persistmempool

Se si salva il mempool quando si spegne e lo si carica al riavvio (default: 1)

-pid=<file>

Specificare il file PID. I percorsi relativi saranno preceduti da un prefisso con una posizione di datadir specifica per la rete. (predefinito: bitcoind.pid)

-punza=<n>

Ridurre le esigenze di stoccaggio permettendo la potatura (rimozione) dei vecchi blocchi. Questo permette di chiamare la catena di potatura RPC per rimuovere blocchi specifici, e permette la potatura automatica dei vecchi blocchi se viene fornita una dimensione target in MIB. Questa modalità è incompatibile con **-txindex** e **-rescan**. Attenzione: Per invertire questa impostazione è necessario scaricare nuovamente l'intera catena di blocchi (default: 0 = disabilita la potatura a blocchi, 1 = consente la potatura manuale tramite RPC, >=550 = potatura automatica dei file di blocchi per rimanere al di sotto della dimensione di destinazione specificata in MIB)

-reindex

Ricostruisce lo stato delle stringhe e l'indice di blocco dei file blk*.dat su disco

-reindex-chainstate

Ricostruire lo stato della catena a partire dai blocchi attualmente indicizzati. Quando si è in modalità di potatura o se i blocchi sul disco possono essere corrotti, utilizzare invece il **reindicatore** completo.

-sighenzia

Creare nuovi file con i permessi di sistema predefiniti, invece di umask 077 (efficace solo con funzionalità di portafoglio disattivate)

-txindex

Mantenere un indice di transazione completo, utilizzato dalla chiamata getrawtransaction rpc (default: 0)

-versione

Versione Print & Go

Opzioni di connessione:

-addnodo=<ip>

Aggiungere un nodo a cui connettersi e cercare di mantenere aperta la connessione (vedere l'aiuto del comando RPC dell'"addendum" per maggiori informazioni). Questa opzione può essere specificata più volte per aggiungere più nodi.

-banscore=<n>

Soglia per la disconnessione dei colleghi che si comportano male (default: 100)

-Nel frattempo...

Numero di secondi per evitare che i colleghi che si comportano male si ricollegino (predefinito: 86400)

-bind=<addr>

Legatevi all'indirizzo indicato e ascoltatelo sempre. Utilizzare la notazione [host]:port per IPv6

-connection=<ip>

Connettiti solo al nodo specificato; **-noconnect** disabilita le connessioni automatiche (le regole per questa coppia sono le stesse di **-addnodo**). Questa opzione può essere specificata più volte per il collegamento a più nodi.

-scopri

Scoprite i vostri indirizzi IP (default: 1 in ascolto e non **-externalip** o **-proxy**)

-dns

Permette ricerche DNS per **-addnodo**, **-semednodo** e **-connect** (predefinito: 1)

-dnsseed

Interrogazione di indirizzi di coppia tramite ricerca DNS, se gli indirizzi sono bassi (default: 1 a meno che non venga utilizzata la **-connessione**)

-enablebip61

Inviare messaggi di rifiuto tramite BIP61 (default: 1)

-externalip=<ip>

Specificare il proprio indirizzo pubblico

-forcednsseed

Interrogate sempre gli indirizzi dei colleghi attraverso la ricerca DNS (default: 0)

-ascolta

Accettare connessioni dall'esterno (default: 1 se non c'è **-proxy** o **-connessione**)

-listenonionion

Creare automaticamente il servizio Tor hidden (predefinito: 1)

-maxconnections=<n>

Mantenere al massimo le connessioni <n> con i colleghi (predefinito: 125)

-maxreceivebuffer=<n>

Massimo buffer di ricezione per connessione, <n>*1000 byte (predefinito: 5000)

-maxsendbuffer=<n>

Massimo buffer di invio per connessione, <n>*1000 byte (predefinito: 1000)

-Impostazione del tempo massimo

Il massimo consentito per la regolazione della compensazione del tempo medio di compensazione delle coppie. La prospettiva dell'ora locale può essere influenzata dalle coppie avanti o indietro di questo importo. (valore predefinito: 4200 secondi)

-maxuploadtarget=<n>

Cercate di mantenere il traffico in uscita sotto il target dato (in MiB per 24h), 0 = nessun limite (default: 0)

-onion=<<<ip:port>

Utilizzare un proxy SOCKS5 separato per raggiungere i peer attraverso i servizi nascosti di Tor, impostare **-noonion** per disabilitare (predefinito: **-proxy**)

-onlynet=<net>

Effettuare le connessioni in uscita solo attraverso la rete <net> (ipv4, ipv6 o cipolla). Le connessioni in entrata non sono interessate da questa opzione. Questa opzione può essere specificata più volte per consentire più reti.

-filtri a coppia

Supporta il blocco del filtraggio e della transazione con i filtri di fioritura (default: 1)

-permitbaremultisig

Relè non P2SH multisig (default: 1)

-port=<port>

Ascolta le connessioni in 'porta' (default: 8333, testnet: 18333, regtest: 18444)

-proxy=<ip:porta>

Connessione tramite proxy SOCKS5, impostare **-noproxy** su off (default: off)

-proxyrandomize

Randomizzare le credenziali per ogni connessione proxy Questo abilita l'isolamento del flusso di Tor (default: 1)

-seednode=<ip>

Collegarsi ad un nodo per recuperare gli indirizzi della coppia e scollegarsi. Questa opzione può essere specificata più volte per il collegamento a più nodi.

-timeout=<n>

Specificare il timeout di connessione in millisecondi (minimo: 1, predefinito: 5000)

-torcontrol=<ip>:<port>:<port>

Porta di controllo Tor da usare se l'ascolto della cipolla è abilitato (default: 127.0.0.0.1:9051)

-password=<pass>

Password della porta di controllo Tor (predefinito: vuoto)

-upnp

Usare UPnP per assegnare la porta di ascolto (default: 0)

-whitebind=<addr>

Collegamento a un certo indirizzo e ai partner nella lista bianca che vi si collegano.
Utilizzare la notazione [host]:port per IPv6

-whitelist=<Indirizzo IP o rete>

Le coppie della lista bianca sono collegate dall'indirizzo IP indicato (ad es. 1.2.3.4) o dalla rete annotata del CIDR (ad es. 1.2.3.0/24). Può essere specificato più volte. Le coppie nella lista bianca non possono essere vietate dal Ministero della Salute

Opzioni di portafoglio:

-tipo di indirizzo

Che tipo di indirizzi utilizzare ("legacy", "p2sh-segwit", o "bech32", default: "p2sh-segwit")

-evitare i costi parziali

Raggruppare le uscite per direzione, selezionando tutte o nessuna, piuttosto che selezionare per ogni uscita. La privacy viene migliorata in quanto un indirizzo viene utilizzato una sola volta (a meno che qualcuno non lo invii dopo che è stato speso), ma può comportare tassi leggermente più elevati in quanto la selezione delle valute può essere subottimale a causa della limitazione aggiunta (default: 0)

-cambiamento del tipo

Quale tasso di cambio utilizzare ("legacy", "p2sh-segwit", o "bech32"). Il valore predefinito è lo stesso di **-addresstype**, eccetto che **-addresstype=p2sh-segwit** utilizza l'output nativo di segwit quando si invia a un indirizzo segwit nativo)

-dal portafoglio

Non caricare il portafoglio e disattivare le chiamate RPC dal portafoglio

-discardfee=<amt>

Il tasso della tariffa (in BTC/kB) che indica la sua tolleranza a scartare la modifica aggiungendola alla tariffa (default: 0,0001). Nota: Un'uscita viene scartata se si tratta di polvere a questo tasso, ma noi scartiamo sempre fino al tasso di ritrasmissione della polvere e un tasso di scarto superiore che è limitato dalla stima del tasso per il target più lungo

-fallbackfee=<amt>

Un tasso di commissione (in BTC/kB) da utilizzare quando la stima della commissione ha dati insufficienti (default: 0,0002)

-keypool=<n>

Impostare la dimensione del pool di chiavi su <n> (predefinito: 1000)

-mintxfee=<amt>

Tassi (in BTC/kB) inferiori a questo sono considerati come tasso zero per la creazione della transazione (default: 0.00001)

-paytxfee=<amt>

Tasso (in BTC/kB) da aggiungere alle transazioni inviate (default: 0.00)

-rescan

Eseguire una nuova scansione della catena di blocco per cercare le transazioni di portafoglio mancanti all'inizio

-salvagewallet

Tentativo di recuperare le chiavi private di un portafoglio corrotto nel bagagliaio

-sviluppo dello scambio di informazioni

Spendere la modifica non confermata al momento dell'invio delle transazioni (default: 1)

-txconfirmtarget=<n>

Se non è stata fissata una tassa di pagamento, includere una tassa sufficiente per iniziare la conferma delle transazioni in media entro n blocchi (default: 6)

-manutenzione del portafoglio

Aggiornare il portafoglio all'ultimo formato all'inizio

-wallet=<percorso>

Specificare il percorso del database del portafoglio. È possibile specificarlo più volte per caricare più portafogli. Il percorso viene interpretato in relazione a <walletdir> se non è assoluto, e verrà creato se non esiste (come una directory contenente un file wallet.dat e file di log). Per la retrocompatibilità, accetta anche i nomi dei file di dati esistenti in <walletdir>).

-walletbroadcast

Effettuare le operazioni di diffusione del portafoglio (default: 1)

-walletdir=<dir>

Specificare la directory per salvare i portafogli (default: <datadir>/portafogli se esiste, altrimenti <datadir>)

-walletnotify=<cmd>

Eseguire il comando quando una transazione di portafoglio cambia (%s in cmd è sostituito da TxID)

-walletrbf

Inviare le transazioni con l'opzione di includere l'intero RBF attivato (solo RPC, default: 0)

-zapwallettxes=<modalità>

Cancellare tutte le transazioni dal portafoglio e recuperare solo le parti della catena di blocco attraverso -rescan all'inizio (1 = conservare i metadati tx-metadati, ad es. informazioni sulla richiesta di pagamento, 2 = rilasciare i metadati tx-metadati)

Opzioni di notifica ZeroMQ:

-zmqpubhashblock=<indirizzo>

Attivare il blocco editoriale in 'indirizzo'

-zmqpubhashblockhwm=<n>

Impostare il blocco di pubblicazione dei messaggi in uscita con una filigrana alta
(predefinito: 1000)

-zmqpubhashtx=<indirizzo>

Attivare la pubblicazione dell'operazione di hashish in 'indirizzo'

-zmqpubhashtxhwm=<n>

Imposta pubblicare il messaggio di uscita della transazione di hashish ad alta filigrana
(default: 1000)

-zmqpubrawblock=<indirizzo>

Attivare il blocco di pubblicazione grezzo in 'indirizzo'

-zmqpubrawblockhwm=<n>

Imposta la filigrana alta del messaggio in uscita del blocco grezzo (predefinito: 1000)

-zmqpubrawtx=<indirizzo>

Attivare la pubblicazione della transazione grezza in 'indirizzo'

-zmqpubrawtxhwm=<n>

Set publish publish gross transaction output message high watermark (default: 1000)

Opzioni di debugging/test:

-debug=<categoria>

Informazioni di debug in uscita (default: **-nodebug**, fornire la "categoria" è facoltativo). Se <categoria> non viene fornita, o se <categoria> = 1, tutte le informazioni di debug vengono emesse. <categoria> può essere: net, tor, mempoolrej, http, bench, zmq, db, rpc, estimatefee, addrman, selectcoins, reindex, cmpctblock, rand, prugna secca, proxy, mempoolrej, libevent, coindb, qt, leveldb.

-debugexclude=<categoria>

Escludere le informazioni di debug da una categoria. Può essere usato insieme a **-debug=1** per generare record di debug per tutte le categorie tranne una o più categorie specificate.

-aiuto-debug

Stampare il messaggio di aiuto con le opzioni di debug e uscire

-logips

Includere gli indirizzi IP nell'output di debug (default: 0)

-logtimestamps

Preparare l'output di debug con la marcatura temporale (default: 1)

-maxtxfee=<amt>

Commissioni totali massime (in BTC) per l'utilizzo su una singola transazione di portafoglio o su una transazione linda; se impostato troppo basso, può annullare le transazioni di grandi dimensioni (default: 0.10)

-stampa per la console

Inviare informazioni di tracciatura/debugging alla console (default: 1 quando non c'è - **daemon**. Per disabilitare la registrazione su file, impostare **-nodebuglogfile**)

-shrinkdebugfile

Ridurre il file debug.log all'avvio del client (default: 1 quando nessun **-debug**)

-uacomment=<cmt>

Aggiungere un commento alla stringa dell'interprete

Opzioni di selezione della catena:

-testnet

Utilizzare la catena di prova...

Opzioni di ritrasmissione del nodo:

-bytespersigop

Equivalenti di byte per sigop nelle transazioni radiotelevisive e minerarie (default: 20)

-datacarrier

Transazioni su supporti dati di relè e di miniera (valore predefinito: 1)

-datacarrierize

Dimensione massima dei dati nelle transazioni dei supporti dati che ritrasmettiamo ed estraiamo (default: 83)

-mempool-sostituzione

Attivare la sostituzione delle transazioni nel pool di memoria (default: 1)

-minrelaytxfee=<amt>

Le commissioni (in BTC/kB) inferiori a questa sono considerate una commissione zero per la ritrasmissione, l'estrazione e la creazione di transazioni (default: 0,000001)

-whitelistforcerelay

Forzare la ritrasmissione delle transazioni dei partner della whitelist, anche se le transazioni erano già in mempool o violano la politica di ritrasmissione locale (default: 0)

-whitelistrelay

Accettare le transazioni trasmesse ricevute dalle coppie della lista bianca anche quando non vengono trasmesse transazioni (default: 1)

Bloccare le opzioni di creazione:

-blockmaxweight=<n>

Impostare il peso massimo del blocco BIP141 (default: 3996000)

-blockmintxfee=<amt>

Impostare il tasso di commissione più basso (in BTC/kB) per le transazioni che sono incluse nella creazione di blocchi. (predefinito: 0.000001)

Opzioni del server RPC:

-Ristorante

Accettare richieste REST pubbliche (default: 0)

-rpcallowip=<ip>

Consentire le connessioni JSON-RPC dalla sorgente specificata. Sono validi per <ip> un singolo IP (ad es. 1.2.3.4), una maschera di rete/rete (ad es. 1.2.3.4/255.255.255.0), oppure una rete/CIDR (ad es. 1.2.3.4/24). Questa opzione può essere specificata più volte

-rpcauth=<userpw>

Nome utente e password HMAC-SHA-256 per le connessioni JSON-RPC. Il campo 'userpw' è disponibile nel formato: 'username': 'SALT': \$ 'HASH'. Un copione di pitone canonico è incluso in share/rpcauth. Il client si connette quindi normalmente utilizzando la coppia di argomenti rpcuser=<NOME UTENTE>/rpcpassword=<PASSWORD>. Questa opzione può essere specificata più volte

-rpcbind=<addr>[:port]

Collegatevi ad un certo indirizzo per ascoltare le connessioni JSON-RPC. Non esponete il server RPC a reti inaffidabili come Internet pubblico! Questa opzione viene ignorata a meno che non venga passato anche **-rpcallowip**. La porta è opzionale e sovrascrive **-rpcport**. Utilizzare la notazione [host]:port per IPv6. Questa opzione può essere specificata più volte (default: 127.0.0.1 e ::1 cioè localhost)

-rpccookiefile=<loc>

Posizione del cookie di autorizzazione. I percorsi relativi saranno prefissati da una posizione di datadir specifica della rete. (predefinito: data dir)

-rpcpassword=<pw>

Password per i collegamenti JSON-RPC

-rpcport=<port>

Ascolta le connessioni JSON-RPC in 'porta' (default: 8332, testnet: 18332, regtest: 18443)

-rpcserialversion

Imposta la serializzazione della transazione grezza o l'esagono di blocco restituito in modalità non verbale, non segwit(0) o segwit(1) (predefinito: 1)

-rpcthreads=<n>

Impostare il numero di thread per gestire le chiamate RPC (default: 4)

-rpcuser=<utente>

Nome utente per le connessioni JSON-RPC

-server

Accettare comandi da riga di comando e JSON-RPC

32. Licenze e utilizzo del software.

Androide

<https://source.android.com/setup/start/licenses>

Termux

<https://github.com/termux/termux-app/blob/master/LICENSE.md>

Nodo

<https://raw.githubusercontent.com/nodejs/node/master/LICENSE>

SQLite

<https://www.sqlite.org/copyright.html>

Git

<https://git-scm.com/about/free-and-open-source>

sqlite-to-rest

<https://github.com/olsonpm/sqlite-to-rest/blob/dev/license.txt>

Redis DB

<https://redis.io/topics/license>

WorldTimeAPI NTP

<http://worldtimeapi.org/pages/faqs#commercial-apps>

Rete Tor

<https://github.com/torproject/tor/blob/master/LICENSE>

Rete di sincronizzazione

<https://forum.syncthing.net/t/syncthing-is-now-mpfv2-licensed/2133>

OpenSSH

<https://www.openssh.com/features.html>

Stucco SSH

<https://www.chiark.greenend.org.uk/~sgtatham/putty/licence.html>

MIT App Inventor 2 Companion e App Inventor Blockly

<https://appinventor.mit.edu/about/termsofservice>

SQLite Expert Personal -freeware

<http://www.sqliteexpert.com/download.html>

Formica Apache

<https://ant.apache.org/license.html>

WGET

<https://www.gnu.org/software/wget/>

OpenJDK

<https://openjdk.java.net/legal/>

Estensioni esterne:

JSONTOOLs

<https://thunkableblocks.blogspot.com/2017/07/jsontools-extension.html>

Per la concessione in licenza delle versioni opensource e commerciali del sistema Mini BlocklyChain consultare il sito ufficiale <http://www.openqbit.com>

Mini BlocklyChain, MiniBlockly, BlocklyCode, MiniBlockMiniChain, QBlockly son marcas registradas por OpenQbit.

Mini BlocklyChain è di pubblico dominio.

Tutto il codice e la documentazione in Mini BlocklyChain è stato dedicato al pubblico dominio dagli autori. Tutti gli autori di codici e i rappresentanti delle società per le quali lavorano hanno firmato delle dichiarazioni giurate che dedicano i loro contributi al pubblico dominio e gli originali di tali dichiarazioni sono conservati in una cassaforte presso gli uffici principali di OpenQbit Mexico. Chiunque è libero di pubblicare, utilizzare o distribuire le estensioni originali Mini BlocklyChain (OpenQbit), sia come codice sorgente che come binari compilati, per qualsiasi scopo, commerciale o non commerciale, e con qualsiasi mezzo.

Il paragrafo precedente si applica al codice e alla documentazione consegnabile in Mini BlocklyChain, quelle parti della libreria Mini BlocklyChain che in realtà raggruppano e spediscono con un'applicazione più grande. Alcuni script usati come parte del processo di compilazione (per esempio, script di "configurazione" generati da autoconf) possono essere inclusi in altre licenze open source. Tuttavia, nessuno di questi script di compilazione lo rende nella libreria finale distribuibile Mini BlocklyChain, quindi le licenze associate a questi script non dovrebbero essere un fattore di valutazione dei diritti di copia e di utilizzo della libreria Mini BlocklyChain.

Tutto il codice di consegna in Mini BlocklyChain è stato scritto da zero. Nessun codice è stato preso da altri progetti o da internet. Ogni riga di codice può essere fatta risalire al suo autore originale, e tutti questi autori hanno dediche di pubblico dominio in archivio. Pertanto, la base di codice Mini BlocklyChain è pulita e non contaminata da codice concesso in licenza da altri progetti open source, non da contributi aperti

Mini BlocklyChain è open source, il che significa che potete fare tutte le copie che volete e fare quello che volete con quelle copie, senza limitazioni. Ma Mini BlocklyChain non è open source. Per mantenere Mini BlocklyChain di pubblico dominio e per garantire che il codice non sia contaminato da contenuti proprietari o concessi in licenza, il progetto non accetta patch di persone sconosciute. Tutto il codice in Mini BlocklyChain è originale, in quanto è stato scritto appositamente per l'uso da parte di Mini BlocklyChain. Nessun codice è stato copiato da fonti sconosciute su Internet.

Mini BlocklyChain è di pubblico dominio e non richiede una licenza. Anche così, alcune organizzazioni vogliono una prova legale del loro diritto di utilizzare Mini BlocklyChain. Le circostanze in cui ciò si verifica sono le seguenti:

- La vostra azienda vuole un risarcimento per le rivendicazioni di violazione del diritto d'autore.
- State utilizzando Mini BlocklyChain in una giurisdizione che non riconosce il pubblico dominio.
- State usando Mini BlocklyChain in una giurisdizione che non riconosce il diritto di un autore di rendere pubblica la sua opera.
- Volete avere un documento legale tangibile come prova che avete il diritto legale di utilizzare e distribuire Mini BlocklyChain.
- Il vostro ufficio legale vi dice che dovete acquistare una licenza.

Se una qualsiasi delle circostanze di cui sopra si applica a voi, OpenQbit, la società che impiega tutti gli sviluppatori di Mini BlocklyChain, vi venderà una garanzia di titolo Mini BlocklyChain. La Garanzia sul titolo è un documento legale che afferma che i pretesi autori del Mini BlocklyChain sono i veri autori, e che gli autori hanno il diritto legale di dedicare il Mini BlocklyChain al pubblico dominio, e che OpenQbit si difenderà con forza contro le rivendicazioni di licenza. Tutti i proventi della vendita delle garanzie di proprietà di Mini BlocklyChain sono utilizzati per finanziare il continuo miglioramento e il supporto di Mini BlocklyChain.

Codice Contributo

Per mantenere Mini BlocklyChain completamente libera e priva di royalty, il progetto non accetta patch. Se volete fare una modifica suggerita e includere una patch come prova del concetto, sarebbe fantastico. Tuttavia, non offendetevi se riscriviamo la vostra patch da zero. Il tipo di licenza non commerciale o opensource che la utilizza in questa modalità e alcune simili senza l'acquisto di supporto sia per uso individuale che aziendale, indipendentemente dalle dimensioni dell'azienda, saranno disciplinate dalle seguenti premesse legali.

Esclusione di garanzia. A meno che non sia richiesto dalla legge applicabile o concordato per iscritto, il Concessore di Licenza fornisce l'Opera (e ciascun Collaboratore fornisce i propri Contributi) "COSÌ COME È", SENZA GARANZIE O CONDIZIONI DI QUAISIASI TIPO, sia espresse che implicite, incluse, senza limitazione, eventuali garanzie o condizioni di TITOLO, NON VIOLAZIONE, COMMERCIALITÀ O IDONEITÀ PER UN PARTICOLARE SCOPO. L'utente è l'unico responsabile della determinazione del corretto utilizzo o della ridistribuzione dell'Opera e dell'assunzione di eventuali rischi associati all'esercizio delle autorizzazioni previste dalla presente Licenza.

Eventuali perdite finanziarie o di altra natura derivanti dall'uso di questo software saranno a carico della parte interessata. Tutte le controversie legali le parti si sotporranno ai tribunali solo nella giurisdizione di Città del Messico, paese Messico.

Per il supporto commerciale, l'uso e la concessione di licenze deve essere stabilito un accordo o un contratto tra OpenQbit o la sua società e la parte interessata.

I termini e le condizioni di marketing della distribuzione possono cambiare senza preavviso, si prega di andare al sito ufficiale www.openqbit.com per vedere eventuali modifiche alle clausole di supporto e di licenza non commerciali e commerciali.

Qualsiasi persona, utente, ente pubblico o privato di qualsiasi natura giuridica o di qualsiasi parte del mondo che utilizzi semplicemente il software accetta senza condizioni le clausole stabilite in questo documento e quelle che possono essere modificate in qualsiasi momento nel portale di www.openqbit.co senza preavviso e possono essere applicate a discrezione di OpenQbit in uso non commerciale o commerciale.

Qualsiasi domanda e informazione su Mini BlocklyChain deve essere indirizzata alla comunità di App Inventor o alle varie comunità del sistema Blockly così come sono: AppBuilder, Trunkable, ecc. e/o alla mail opensource@openqbit.com per la richiesta di domande può prendere la risposta da 3 a 5 giorni lavorativi.

Supporto con uso commerciale.

support@openqbit.com

Vendite per uso commerciale.

sales@openqbit.com

Informazioni legali e domande o dubbi sulla licenza

legal@openqbit.com

